DES - TEF

Diplôme d'Études Spécialisées en

Technologie de l'Éducation et de la Formation

Année académique 2002 – 2003

Facultés Universitaires Notre-Dame de la Paix – Namur

Université de Liège

Mémoire

Apprendre à programmer
Pourquoi ?
Comment ?
CAR : Computers Are Robots

Rupert Meurice de Dormale

Apprendre à programmer
Pourquoi ?
Comment ?
CAR : Computers Are Robots

Préliminaire

Ce mémoire est réalisé dans le cadre de la certification, pour l'année académique 2002-2003, du Diplôme d'Études Spécialisées en Technologie de l'Éducation et de la Formation organisé conjointement par l'Université de Liège (ULg) et les Facultés Universitaires Notre-Dame de la Paix à Namur (FUNDP).

Son objet est la description d'une nouvelle méthode d'apprentissage des bases de la programmation destinée aux débutants adolescents et adultes. Pour la facilité de l'exposé, une dénomination a été donnée à cette méthode : CAR → Computers Are Robots.

En annexe de ce mémoire se trouvent :

1. Un cédérom reprenant l'ensemble de cette méthode à son stade actuel de développement
2. Le mode d'emploi de ce cédérom
3. Le mode d'emploi (non encore implémenté) du module 5 (robotique virtuelle)
4. Le mode d'emploi (non encore implémenté) du module 8 (programmation virtuelle en pseudocode)
5. Le texte de la communication « *Robotique virtuelle : environnement pour un apprentissage dynamique et interactif de l'algorithmique* » faite au 5ème colloque international de Robotique pédagogique à Montréal en 1997.
6. Les changements pédagogiques induits sont expliqués au travers d'une interview de l'auteur par Pierre, Adèle et Amaury
7. Le logiciel sous toutes ses coutures au travers d'une analyse menée par Michel
8. Et puis l'avis des élèves dans tout ça ??? !!!

Introduction

Ce mémoire se voulait d'envergure, à savoir la validation d'une méthode d'apprentissage des bases de la programmation destinée au débutant absolu en la matière, validation se trouvant au carrefour de sept axes :

1. La question de savoir si l'apprentissage de la programmation représente un intérêt quelconque pour un étudiant qui ne se destine pas à ce métier
2. Une méthode préexistante sur laquelle se base et que complète la méthode présentée
3. Une expérience de plus de vingt ans dans l'enseignement de la programmation à des débutants
4. La littérature relatant les difficultés des novices dans l'apprentissage de ce savoir-faire
5. Les nombreuses méthodes mises au point pour faciliter cet apprentissage
6. Les changements pédagogiques induits par l'innovation
7. Les aspects techniques et technologiques de l'innovation

Malheureusement, par manque de temps et de moyens, ce projet subira quelques amputations :

♦ Le point 1 ne sera pas abordé. Au niveau des « bienfaits de l'apprentissage de la programmation », l'examen de la littérature de la fin des années 80 soit avance des suppositions non étayés, soit relate des expériences discutables sur les transferts de ce genre d'apprentissage (Romainville). Un temps considérable devrait être consacré à la recherche d'articles plus récents, censés être parus au cours d'une période sans doute moins prolifique durant laquelle la programmation pure et dure à cédé du terrain à l'utilisation des progiciels d'abord, au multimédia ensuite, et à l'exploitation de l'Internet enfin.

- Le point 5 ne sera pas abordé non plus. Les années 90 ont vu éclore des méthodes d'aide à l'apprentissage de la programmation permettant au novice de franchir telle ou telle étape difficile du cursus[1]. Comme pour le point 1, du temps devrait être consacré afin de recenser les méthodes apparues dernièrement et vérifier s'il n'y a pas double emploi avec la méthode préconisée.
- Le point 4 se résume à une trop brève incursion dans l'univers pléthorique des publications consacrées aux problèmes que rencontrent les débutants ou novices dans l'apprentissage de la programmation
- L'expérience de l'auteur a donné lieu à un certain nombre d'assertions résultant d'observations, d'intuitions, d'intimes convictions qui seront prises en compte dans cet exposé, même si certaines d'entre elles n'ont pas trouvé d'écho dans la littérature ou n'ont pas fait l'objet d'une validation expérimentale.
- Les points 6 et 7 seront abordés au niveau de témoignages en annexe.

Ces restrictions étant faites, ce mémoire se déploiera selon l'architecture suivante :

- Le but exact de la méthode proposée
- La présentation de la méthode de base (Images pour programmer) et en quoi elle constituait une approche totalement novatrice et visionnaire dans l'apprentissage de la programmation
- Un bref historique des faits qui ont amené l'auteur à prolonger cette méthode de base par l'implémentation de certains modules et, finalement, par la conception d'une méthode complète assistée par ordinateur.
- L'organisation de la nouvelle méthode
- Qu'est devenue la nouvelle méthode par rapport à « Images pour programmer »
- Le détail de chacun des modules de la nouvelle méthode, ses objectifs, ses motivations au regard de l'expérience accumulée et de la littérature consultée, les avantages et inconvénients de son implémentation et des conseils méthodologiques s'y rapportant
- L'apport du DES-TEF dans l'évolution de la méthode
- Et pour terminer, une micro-fiche technique du logiciel.

But de la méthode CAR

Par « apprendre à programmer », il faut entendre la maîtrise d'un savoir-faire consistant à spécifier, rédiger, faire exécuter, tester et débugger des programmes informatiques en réponse à des demandes (énoncés) plus ou moins imprécises. Dans ce cas, l'ordinateur est utilisé comme l'exécutant des programmes que l'apprenant a rédigés, et non comme un outil pour réaliser des tâches telles que du traitement de texte, des feuilles de calcul électroniques, la gestion de bases de données, etc.

De plus, si, dans le cursus, certains outils et métaphores sont utilisés pour faciliter l'apprentissage du novice en la matière, l'objectif final est qu'il sache, avec son crayon et face à sa feuille de papier, rédiger les programmes adéquats répondant aux énoncés qui lui sont soumis sans que, dans cette phase de conception, le recours à la machine lui soit nécessaire.

Cette finalité doit être gardée à l'esprit, surtout pour les pédagogues qui utiliseraient la méthode pour enseigner la programmation à leurs étudiant(e)s.

Il s'agit bien d'une *méthode* et non d'une succession de « trucs et astuces ». L'ensemble du curriculum est le résultat d'une réflexion pédagogique destinée à conduire le novice absolu, qui n'a pas encore idée de ce que veut dire le mot « programmer », à la réalisation de programmes gérant les variables indicées et les

[1] Nos recherches nous ont mené à la découverte de quelques méthodes proposées pour aider le débutant à franchir l'un ou l'autre cap difficile de la programmation. FPL propose la conception des programmes sous forme de « flow-charts ». Le Stanford BIP Project montre le contenu des variables et offre une assistance « intelligente » au débuggage des programmes, etc.

procédures paramétrées en Pascal (ou en tout autre langage de type procédural). Un élève motivé peut trouver dans cette méthode la source de plus d'une centaine d'heures de travail.

Programmer représente une difficulté majeure pour le débutant… et plus encore pour l'enseignant qui se voit investi de la tâche d'aider à cet apprentissage.

La base : « *Images pour programmer* »

Dans le début des années 90, l'essentiel de la pédagogie consistait à apprendre directement un langage au débutant, lui expliquer les effets de chaque instruction et lui montrer le fonctionnement de programmes simples. Il ne lui restait plus alors qu'à « faire le reste ».

Manifestement, cela ne fonctionnait pas. L'importance de la littérature sur le sujet le confirme. De manière assez étonnante, de nombreux constats d'échec sont dressés, mais peu de remèdes concrets sont proposés. Ceux-ci ont la particularité d'être ponctuels, de s'attacher à aider au passage d'une difficulté bien précise sans revoir le problème dans son ensemble. D'une manière générale, du Boulay et al. attire l'attention sur l'importance de la « notional machine » (que l'on pourrait traduire par « machine fonctionnelle ») qui se cache derrière chaque langage et de l'intérêt qu'il y a à expliquer ce concept aux débutants, mais sans toutefois concrétiser son idée de façon précise[2].

C'est dans ce contexte que voit le jour « Images pour programmer »[3].

Cette méthode d'apprentissage des bases de la programmation est révolutionnaire à plus d'un titre :

- L'approche du sujet est repensée de fond en comble
- La méthode est simple, complète (manuel + séquences vidéo)[4] et cohérente
- La méthode peut facilement être mise en œuvre et donne d'excellents résultats pour les adultes.

Le principe de base est de fournir au débutant une représentation simple et imagée de la manière dont cette boîte noire que représente l'exécutant-ordinateur fonctionne et qu'il puisse s'appuyer sur cette « vision » mentale pour concevoir ses programmes.

Le monolithe auquel devait faire face le débutant lors de son étude de la programmation au travers de la syntaxe est maintenant taillé de marches permettant un franchissement plus aisé.

---

[2] du Boulay et al. signale toutefois que « *pour aider à apprendre ses propriétés, la machine fonctionnelle doit être simple, constituée d'un petit nombre de constituants interagissant d'une façon qui doit être facilement compréhensible par le novice, si possible par analogie avec d'autres mécanismes avec lesquels il est plus familier. Des méthodes doivent également être fournies au novice pour observer certains de ses travaux en action.* ». L'auteur met en garde contre une simplicité superficielle de la machine fonctionnelle, simplicité qui peut être atteinte au détriment de la qualité et qui risque de déboucher sur une difficulté plus grande d'exploitation par le novice (ambiguïtés, pauvreté sémantique des primitives,…).

[3] Texte intégral consultable sur http://www.det.fundp.ac.be/cefis/publications/doconline.html

[4] Voir description dans Haugulstaine-Charlier B.

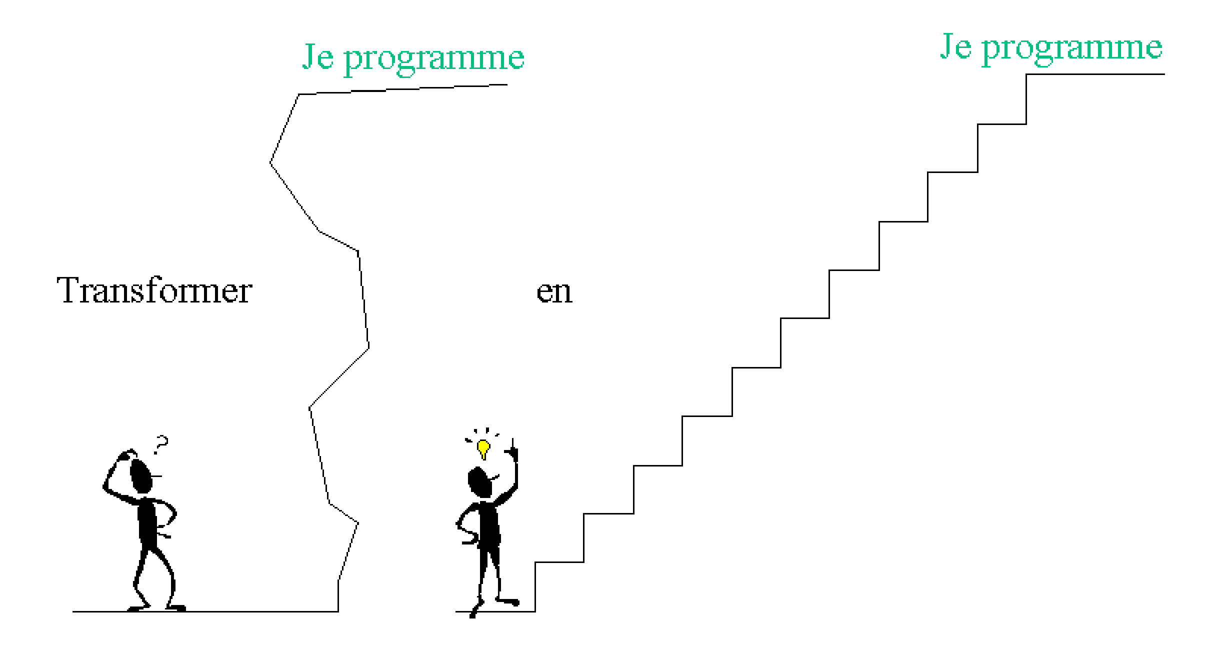


Figure 1

« Images pour programmer » s'attache d'abord à définir ce qu'est programmer et les constituants d'un programme. L'algorithmique est alors abordée au travers de la programmation de petits robots. Vient ensuite une description de l'exécutant-ordinateur (gestion des « casiers » (variables), communication avec l'extérieur, manipulation de l'information et compréhension des structures de contrôle). Un certain nombre de « tours de main » sont abordés en pseudocode (langage "naturel" de description d'algorithme) et en GNS[5]. Ce n'est qu'après cette solide « introduction » que la syntaxe du langage (Pascal) est abordée.

Cette méthode prenait donc l'exact contre-pied des pratiques de l'époque. Les principes de la programmation sont abordés sur base d'une machine fonctionnelle imagée, claire et simple. La syntaxe n'apparaît plus alors que comme une simple traduction permettant l'encodage du programme. La majeure partie de l'énergie dépensée par l'apprenant est investie dans « l'invariant de la programmation » (l'analyse du programme proprement dite) pour ne laisser que peu de place à la syntaxe, différente selon les langages.

Au vu de tout ce qui précède, nous sommes surpris que cette méthode n'ait pas eu le retentissement international qu'elle méritait.

Historique de la méthode CAR

En tant qu'élève du Professeur Duchâteau, j'ai appliqué directement « Images pour programmer » dans le cours de programmation dont je venais d'être fraîchement investi.

Comme il est dit plus haut, cette méthode fonctionne et donne d'excellents résultats pour les débutants adultes. Il s'est toutefois avéré qu'environ un tiers des élèves adolescents (16-18 ans) présentaient des difficultés au moment de la rédaction de leurs premiers programmes.

---

[5] GNS = graphes de Nassi-Schneidermann permettant de mieux visualiser le flux d'exécution du programme. Ceci est corroboré par Cunniff et al. qui signalent des améliorations notables chez les débutants qui utilisent leur langage FPL (First Programming Language) se présentant sous forme de « flow-chart ». Dans ce système, les structures de contrôles sont visuellement isolées, ce qui oblige l'apprenant à déterminer explicitement les instructions qui doivent prendre place dans ces « digressions ».

Ces difficultés m'ont poussé à introduire des travaux pratiques en Logo sur les répétitives imbriquées (dessins de rosaces) au moment où les structures de contrôle étaient abordées, ce qui semblait aider les élèves dans une certaine mesure.

Après cette étape, les exercices des petits robots étaient réalisés, et ceci avec des classes comptant de vingt à vingt-cinq élèves. Face à un tel nombre, la correction des exercices était un réel problème, surtout quand il fallait expliquer son erreur à chaque élève.

Des cours de rattrapage et des corrections individuelles furent alors mises sur pied, mais il s'est avéré que les explications verbales, abstraites du fonctionnement des structures de contrôle ne portaient que peu de fruit. En outre, le suivi des élèves ayant vraiment des grosses difficultés a révélé que leurs problèmes relevaient essentiellement d'une mauvaise compréhension du flux d'exécution du programme au niveau des structures de contrôle.

Face à ces différents problèmes, l'idée fut envisagée que l'élève puisse programmer et tester seul, dans une environnement interactif, ses programmes sur les petits robots. Il pourrait ainsi se construire une représentation correcte du fonctionnement des structures de contrôle à travers ses essais et erreurs, comme c'est le cas dans l'environnement Logo.

Une première version du module de robotique virtuelle (incluant les 4 robots de « Image pour programmer » + 1 robot personnel) fut alors réalisée en 1995 et testée sur des élèves. L'interface de construction des programmes manquant d'ergonomie, une nouvelle version (la version actuelle) de 17 nouveaux robots de difficulté progressive fut réalisée (voir le module 5 « robotique virtuelle » ci-dessous).

La proposition d'écrire deux articles[6] sur ce module m'a obligé à examiner de plus près les raisons pour lesquelles les élèves utilisant cet environnement faisaient des progrès importants dans la maîtrise des structures de contrôle. En résumé, le problème vient du fait que les débutants adolescents ne possèdent pas encore un sens de l'abstraction suffisamment développé pour gérer correctement le flux des programmes de façon totalement abstraite. Il s'avérera par la suite que ce problème d'abstraction est en fait le nœud gordien des problèmes de l'apprentissage de la programmation.

La mise à la disposition des élèves de ce module eut un « effet secondaire » inattendu. Les élèves travaillant en autonomie ont rapidement montré des différences importantes dans leurs vitesses de progression, posant de sérieux problèmes à la reprise d'un cours ex cathedra par la suite.

Vint alors l'idée d'implémenter le cours complet sur ordinateur tout en exploitant les nombreuses possibilités du multimédia. Cette implémentation fut réalisée en juillet et août 2000.

Le didacticiel ainsi créé fut utilisé dès la rentrée de septembre pour l'ensemble des classes. Cette utilisation permit de constater une nette amélioration au niveau de l'algorithmique, mais des problèmes subsistaient au niveau de la réalisation des programmes informatiques en pseudocode.

Le module 8 fut alors développé en février 2002. Sa structure est très voisine du module 5 sur la robotique virtuelle. Ce module simule le fonctionnement de l'ordinateur lors de l'exécution du programme, montrant les échanges avec le clavier et l'écran et la gestion des variables. Ce module, en « imageant » la machine fonctionnelle, a permis une fois de plus aux élèves de surmonter leurs difficultés.

Au bout de cinq ans d'utilisation et d'amélioration de cette nouvelle méthode, je peux dire que les progrès sont surprenants et que plus aucun élève ne présente de difficultés majeures dans la réalisation de programmes simples, que ce soit en pseudocode ou en Pascal. Il s'avère toutefois que les difficultés se sont reportées plus loin, au niveau de l'utilisation des variables indicées et des procédures paramétrées.

---

[6] Pour les détails, voir Meurice de Dormale R. 1996 et 1997 (en annexe)

J'ai donc le projet de développer deux modules permettant aux élèves de visualiser ce qui se passe lors de l'exécution des programmes concernant ces deux derniers points.

Organisation de la méthode CAR

Lorsqu'un novice apprend à programmer, son problème est de surmonter simultanément trois difficultés :

1. La maîtrise de l'algorithmique (notions abstraites)
2. La maîtrise de la gestion des variables (notions abstraites)
3. La fusion des deux éléments précédents afin de pouvoir écrire des programmes qui vont traiter (de façon abstraite dans une machine abstraite) de « l'information » afin de fournir des résultats corrects quelles que soient les données introduites dans la machine.

Comme le montre la figure 1, ces trois difficultés cumulées se révèlent infranchissables. L'idée est donc de découper le cursus en étapes sans qu'aucune de celles-ci ne présente subitement l'introduction importante d'éléments soit trop complexes, soit trop nouveaux. Idéalement, le passage d'un module à l'autre devrait être quasiment insensible.

En outre, ce ne sont pas les contenus qui sont le plus à craindre. Les jeunes débutants sont confrontés à une difficulté de taille : l'abstraction, omniprésente dans le domaine qui nous occupe.

Même si, de façon quelconque, une méthode « concrète » permet à un débutant de réaliser ses premiers pas en faisant appel à ses ressources, à ce qu'il connaît déjà, en se référant à du concret, tôt ou tard, un saut dans l'abstrait se produit. C'est à ce moment que « ça passe ou ça casse ».

La tentative de cette nouvelle méthode est d'établir une « passerelle avec main courante » entre les deux falaises afin que ce saut ne doive jamais avoir lieu et que le plus grand nombre puisse atteindre l'autre bord. En d'autres termes, il faudrait faire en sorte qu'à l'apparition d'une nouvelle situation abstraite, le débutant ait comme ressource, d'une part, les images d'une situation semblable concrète qu'il a déjà eu à résoudre et, d'autre part, les images de la machine fonctionnelle qu'il utilise afin de concevoir son programme. On diminue ainsi d'autant le seuil à franchir pour satisfaire à la tâche demandée. Progressivement, le débutant se constituera ainsi un stock d'images adéquates et un stock de stratégies à mettre en œuvre dans tel ou tel cas qui lui serviront de support à la résolution de problèmes nouveaux.

Vue de l'extérieur, la méthode est organisée de la façon suivante :

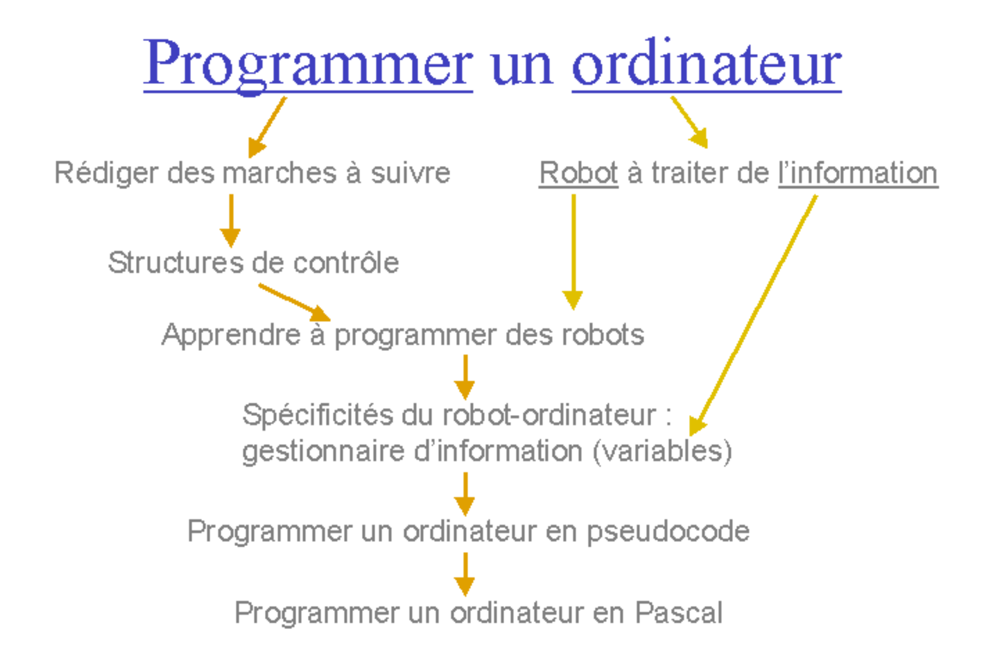


Figure 2

Un premier saut dans l'abstrait peut déjà avoir lieu au moment de maîtriser les structures de contrôle (conditionnelle, répétitive). Une passerelle consiste en la mise en place d'un environnement d'expérimentation qui permet au débutant de construire ses concepts de fonctionnement des structures de contrôle par essai et erreurs tout en faisant appel à des situations (concrètes) connues qui ne le coupent pas de ses ressources (pour les détails, voir article « Robotique virtuelle : un environnement pour un apprentissage dynamique de l'algorithmique » en annexe).

Plus loin, n'en déplaise aux puristes, un triple saut peut être évité en optant résolument pour l'option selon laquelle « **les ordinateurs sont des robots** »[7].

Si on conserve une distinction entre les deux types de machines, il est nécessaire d'introduire de nouvelles spécifications lorsqu'on passe des robots aux ordinateurs, introduisant une certaine cassure dans le cursus. Cette cassure nécessite l'injection d'une quantité de matière et de nouveautés substantielles. En revanche, l'amalgame proposé permet d'obtenir un continuum : le passage du robot à l'ordinateur ne nécessite plus, de ce fait, que l'énoncé de ses spécificités propres, comme on aurait à le faire pour un robot trieur d'enveloppes ou gestionnaire de stock.

De manière plus détaillée, ce continuum peut se concevoir de la manière suivante :

- L'ordinateur est un robot, il se définit donc, comme n'importe quel robot, par
  - Les actions élémentaires qu'il peut effectuer
  - Les conditions qu'il peut tester
  - Les structures de contrôle qu'il peut exécuter
- L'ordinateur est un robot ayant les spécificités suivantes :
  - Il reçoit de l'information de l'extérieur
  - Il manipule cette information (à l'intérieur)
  - Il fournit de l'information vers l'extérieur

Pourquoi cette vision des choses évite-t-elle un triple saut dans l'abstrait ?

Premièrement, du fait même du continuum énoncé ci-dessus, il n'est plus question de s'arrêter sur la voie des robots pour repartir sur l'étude de l'ordinateur nécessitant la mise en œuvre de nouveaux concepts.

Deuxièmement, cela permet d'éviter que l'introduction de la notion de variable apparaisse ex abrupto, comme c'est souvent le cas. Cela représente pour le débutant un choc frontal avec « quelque chose » de totalement inconnu. L'amalgame robot-ordinateur permet d'introduire ce concept de façon douce en proposant au débutant, dans le module 5 sur la robotique virtuelle, de gérer des robots qui possèdent des ardoises afin d'y noter certains renseignements. Au moment où la notion abstraite de variable sera abordée, les stratégies de son « jumeau concret » (la manipulation de l'ardoise) seront déjà en place. L'introduction des variables grâce à l'analogie des planches à autocollants (voir module 6) sera « indolore », le concept général et les habitudes de manipulation des données étant déjà présentes.

Troisièmement, les petits modules de programmes gérant des tâches fréquentes (compteurs, sommes, etc.) appelés tours de main dans « Images pour programmer » pourront être introduits en robotique virtuelle. C'est déjà le cas pour trois d'entre eux (deux compteurs et une somme). Ce sera le cas prochainement pour tous les tours de main vus dans la méthode (par exemple la position du premier et du dernier qui sera introduite par un robot possédant des ardoises et dont la tâche sera de signaler les positions de la première et de la dernière voiture rouges qui passent devant la fenêtre). Cette tactique permettra d'éviter la « double difficulté » : que le débutant soit confronté à une stratégie à mettre en place d'une part, qui plus

---

[7] Si on se place du côté du débutant, d'autres avantages émanent de cette affirmation. D'un côté, elle possède quelque chose de rassurant : le débutant a le sentiment de s'introduire dans un univers moins inconnu, ou, à tout le moins, dans un univers qui fait partie de son imaginaire depuis longtemps. D'un autre côté, cela démythifie le côté « intelligent » de la machine, un robot ne réalisant que les actions qui lui sont commandées. Le départ se fait donc sur des bases plus saines.

est dans un univers abstrait qu'il connaît mal (ou qu'il ne connaît pas) d'autre part ? En revanche, si, préalablement, le débutant a dû développer une stratégie face à un problème d'inspiration concrète (positions de la première et de la dernière voitures rouges), il aura moins de mal à gérer le problème plus abstrait qu'est le fait de signaler les positions du premier et du dernier mots « attention » dans une liste de mots, qui plus est, quand cette liste des mots est « n'importe laquelle » passant par la tête d'un utilisateur inconnu.

Après ces introductions aux variables (module 6) et aux tours de main (module 7), l'apprenant évite un nouveau saut et retrouve un univers connu : un module de robotique virtuelle spécialement dédicacé à l'ordinateur (voir module 8) disposant de toutes les instructions nécessaires à la conception de petits programmes en pseudocode. Huit variables y sont disponibles ainsi qu'un écran et un clavier. L'apprenant peut utiliser ce module pour réaliser un grand nombre de programmes simples, y compris des programmes utilisant le tirage au hasard. L'exécution de tous les programmes se fait sous forme d'animations qui montrent comment les choses se passent, tant à l'intérieur qu'à l'extérieur de l'ordinateur.

Vient enfin l'introduction de la syntaxe qui, comme il a été dit plus haut, ne joue qu'un rôle franchement secondaire dans le curriculum ; on peut même envisager que les programmes conçus en pseudocode soient traduits automatiquement en Pascal (ou en tout autre langage de type procédural). Il ne faut toutefois pas perdre de vue que l'objectif est de rendre l'apprenant autonome face à une feuille blanche.

La méthode atteindra ses objectifs finaux quand deux modules de robotique virtuelle adaptés aux variables indicées et aux procédures paramétrées seront implémentés.

Différences entre CAR et *Images pour programmer*

Sur un grand nombre de points, la méthode CAR n'apporte fondamentalement rien de neuf par rapport à « Images pour programmer ».

Les concepts, images et analogies sont généralement conservés.

L'adoption d'un support interactif permet toutefois l'apport d'éléments supplémentaires tels que les animations, les auto-tests, etc.

Cette interactivité devient particulièrement intéressante de par la possibilité de développer des univers d'expérimentation. L'expérience montre que, pour les personnes ayant quelques difficultés d'abstraction, ces univers d'expérimentation sont particulièrement bienvenus.

C'est le cas pour le module 5; l'environnement de robotique virtuelle permet à l'apprenant, par essais et erreurs, de se forger une représentation solide et correcte de la façon dont fonctionnent les structures de contrôle, sésame indispensable pour aborder la programmation.

Une légère inflexion est donnée quant au passage des robots aux ordinateurs, la méthode CAR essayant d'estomper au maximum les différences entre ces deux machines. Ceci permet d'utiliser l'environnement de robotique virtuelle pour introduire plus aisément certains concepts abstraits. Ainsi, le module 5 devient un « camp de base » permettant de soutenir le débutant dans ses « expéditions » "*variables*", "*tours de main*" et "*programmation en pseudocode*". Cette programmation en pseudocode peut alors être abordée dans un environnement d'expérimentation tout à fait semblable à celui des robots virtuels.

Il est toutefois à regretter que certaines richesses de « Images pour programmer » aient été (provisoirement) délaissées.

- ♦ C'est notamment le cas de la démarche descendante et structurée dont l'approche procédurale est l'outil privilégié; il est indispensable de promouvoir cette approche dès les premiers exercices.
- ♦ Il serait judicieux que les concepts de « Quoi Faire Faire » et « Comment Faire Faire » retrouvent une place de choix dans la méthode.

- L'absence de GNS, permettant au débutant de visualiser le flux d'exécution des programmes, est également regrettable.

Ces lacunes seront comblées dans les mois à venir.

CAR module par module

Par souci de brièveté, seuls les modules apportant quelques nouveautés sont développés en détail, à savoir les modules 4, 5, 6, 7 et 8 (signalés en bleu). Les autres sont survolés pour mémoire. Le lecteur intéressé trouvera l'entièreté de la méthode sur le cédérom joint.

Module 0 : le contrat

Il s'agit du contrat passé entre « la méthode » et l'apprenant. L'apprenant est informé des difficultés qui l'attendent. Il est également mis au courant des bénéfices qu'il pourra retirer de son apprentissage[8]. Actuellement, seuls les arguments de rigueur du langage et de la pensée sont avancés.

Notre intime conviction nous porte à croire, en tout cas pour certains individus, que la pratique de la programmation procure ou développe des capacités à résoudre des problèmes et à construire des projets. Mais les travaux de Romainville nous poussent à une grande prudence quant aux affirmations sur les intérêts et les transferts que pourrait procurer l'apprentissage de la programmation. L'examen de la littérature récente serait indispensable pour faire le point à ce niveau.

Citons toutefois Linn M.C. et Dalbey J. qui considèrent que, pour autant que l'apprentissage de la programmation ait été fait en classe et avec un feed-back riche, les élèves développent des capacités explicite à former des modèles et à travailler de façon modulaire ou procédurale.

L'examen des travaux de Gagné nous semble un piste très intéressante au niveau de l'apprentissage des concepts, principes et solutions de problème… à suivre !

Module 1 : présentation succincte des modules et de leur enchaînement

Le but est de montrer la cohérence de la méthode afin que le débutant ne butine pas de chapitre en chapitre sans ordre précis ni sans comprendre que la maîtrise d'un module est indispensable pour passer au module suivant.

Module 2 : qu'est-ce que programmer ?

Ce module explique que programmer est de l'ordre du « Faire Faire » et des conséquences qui en découlent. Ce chapitre attire l'attention du débutant sur toutes les caractéristiques d'un programme. La différence fondamentale entre un humain (qui peut combler des imprécisions) et la machine est alors illustrée. Les aspects à maîtriser pour programmer sont ensuite énoncés : connaître l'exécutant, connaître la tâche à réaliser, rédiger le programme et ensuite le valider pour toutes les conditions de fonctionnement imaginables. Pour terminer, un auto-test permet à l'apprenant de se situer au niveau de ses acquis.

Module 3 : qu'est-ce qu'un programme, de quoi est-il constitué ?

Le module va décomposer la marche à suivre en ses différents composants afin d'introduire les structures de contrôle et faire apparaître leur importance.

---

[8] Dans son relevé sur les 5 difficultés majeures dans l'apprentissage de la programmation, du Boulay signale en premier lieu un problème d'orientation générale : la question de savoir à quoi est destinée la programmation, quel genre de problèmes peuvent être abordés et quels sont les avantages éventuels des efforts dépensés pour acquérir la compétence. En plus d'insister sur les avantages, il nous paraît intéressant que ce chapitre 0 aborde à l'avenir les deux autres aspects.

Module 4 : particularités des structures de contrôle

L'objectif du module est d'apprendre au débutant les principales structures de contrôle, leurs particularités, leurs fonctionnements de leurs mises en œuvre. Un chapitre est consacré à la construction des expressions booléennes et à leur négation (théorème de de Morgan). Cette construction des conditions est souvent négligée dans les méthode d'apprentissage alors que Spohrer constate, au cours d'une de ses expérimentations, que 48% des bugs relevés proviennent d'une mauvaise construction des conditions des structures de contrôle, notamment par confusion entre le ET et le OU lors de la négation de conditions. Il recommande aussi l'étude du théorème de de Morgan[9].

Chaque structure de contrôle est ensuite mise en œuvre dans un programme que l'apprenant peut faire exécuter pas à pas. Il peut alors voir, à chaque étape, ce qui se passe à l'extérieur de l'exécutant (lorsqu'une instruction engendre une action de l'exécutant), mais aussi, et surtout, ce qui se passe à l'intérieur de l'exécutant lorsqu'un élément d'une structure de contrôle se présente, particulièrement quand c'est le test d'une condition[10].

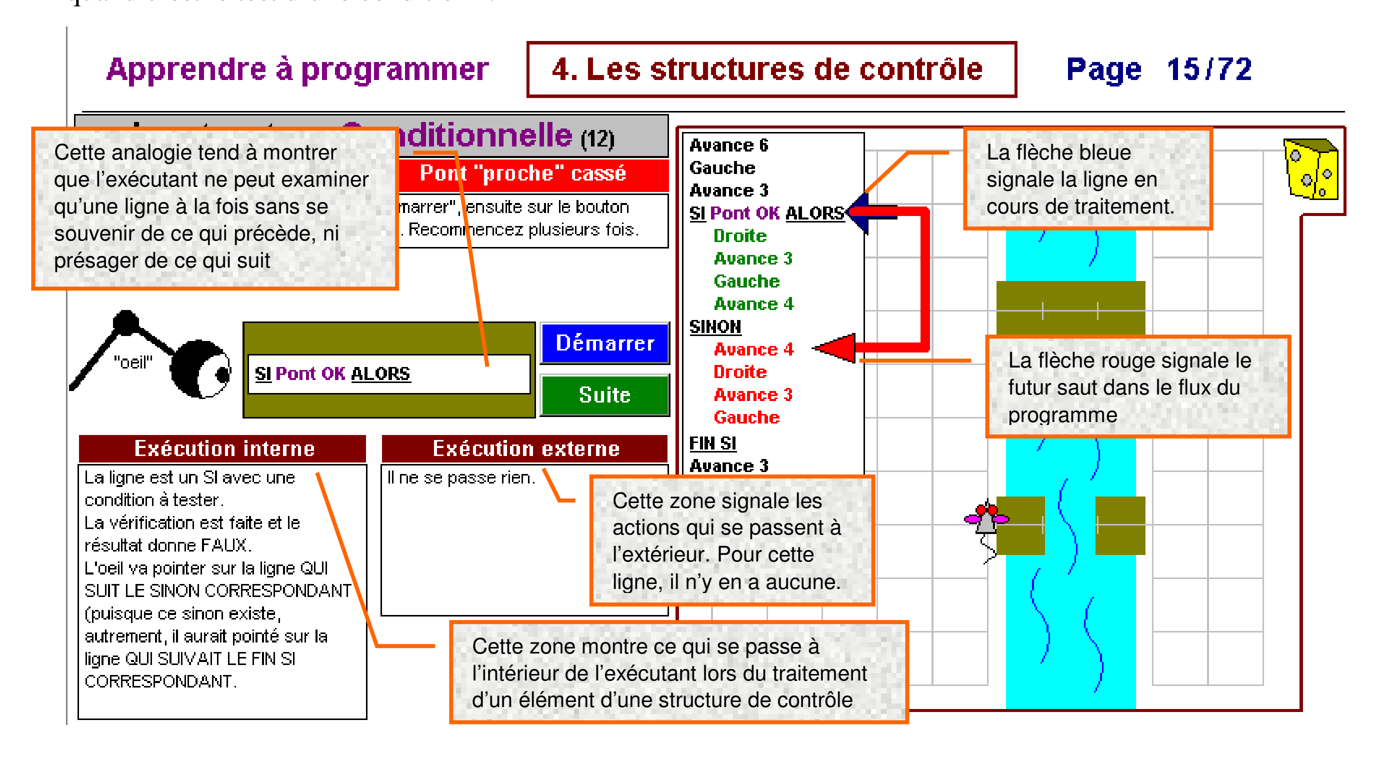


Figure 3

À ce niveau, on peut parler d'une phase d'imprégnation : l'apprenant observe l'exécution du programme et les réactions extérieures engendrées chez l'exécutant. Il peut aussi observer les réactions intérieures provoquées par les structures de contrôle et la suite que cela engendrera dans l'exécution du programme.

---

[9] Théorème de de Morgan : la complémentation d'un OU devient un ET avec chaque terme complémenté (herbe mouillée si il pleut **OU** si j'arrose. Herbe PAS mouillée si il ne pleut PAS **ET** si je n'arrose PAS). La complémentation d'un ET devient un OU avec chaque terme complémenté (je peux conduire si j'ai le permis **ET** si j'ai la clé. Je ne peux pas conduire si je n'ai PAS le permis **OU** si je n'ai PAS la clé).

[10] À notre avis, pour le peu de littérature consultée, cet aspect « s'il ne se passe rien dehors, c'est qu'il se passe quelque chose dedans » est neuf et semble s'avérer bénéfique pour le débutant qui peut ainsi prendre conscience qu'il y a, en quelque sorte, deux exécutants : l'automate qui exécute les actions extérieures et, à l'intérieur de celui-ci, un second automate qui possède un programme pour exécuter le programme (en gérant les structures de contrôle, testant les conditions, effectuant les branchements,…). Cet aspect est à approfondir et pourrait fournir des retombées didactiques intéressantes.

D'autre part, il faut que le débutant comprenne rapidement qu'un programme ne doit pas fonctionner que dans un cas précis, mais dans tous les cas possibles et imaginables. L'attention du novice est attirée sur ce point notamment par la présentation d'un même programme s'exécutant dans des circonstances différentes tout en montrant son résultat au niveau de l'exécution externe, comme c'est le cas dans la figure ci-dessous.

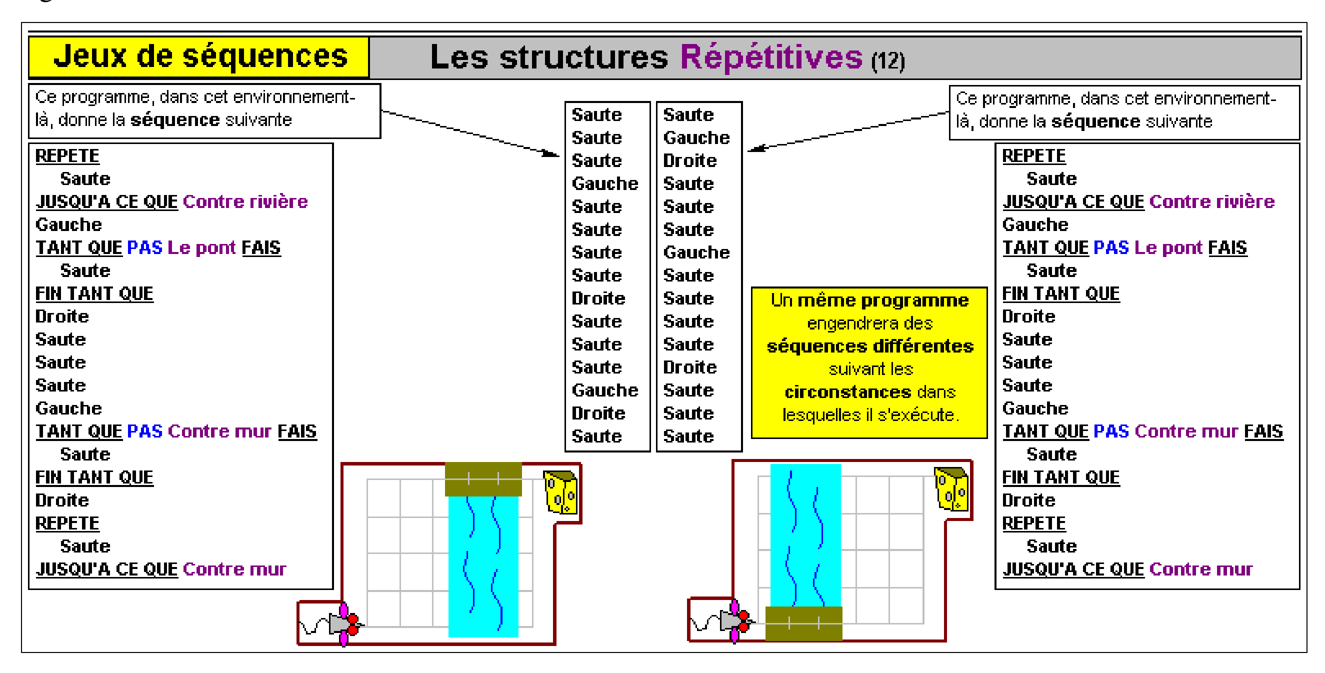


Figure 4

Après cette phase, un grand nombre de questions (28) sont posées afin que l'apprenant puisse vérifier ses acquis. Ces questions se divisent en deux grands groupes :

1. Les questions plus ponctuelles demandant explicitement à quelle ligne du programme va sauter l'exécution en fonction des conditions présentées dans l'énoncé

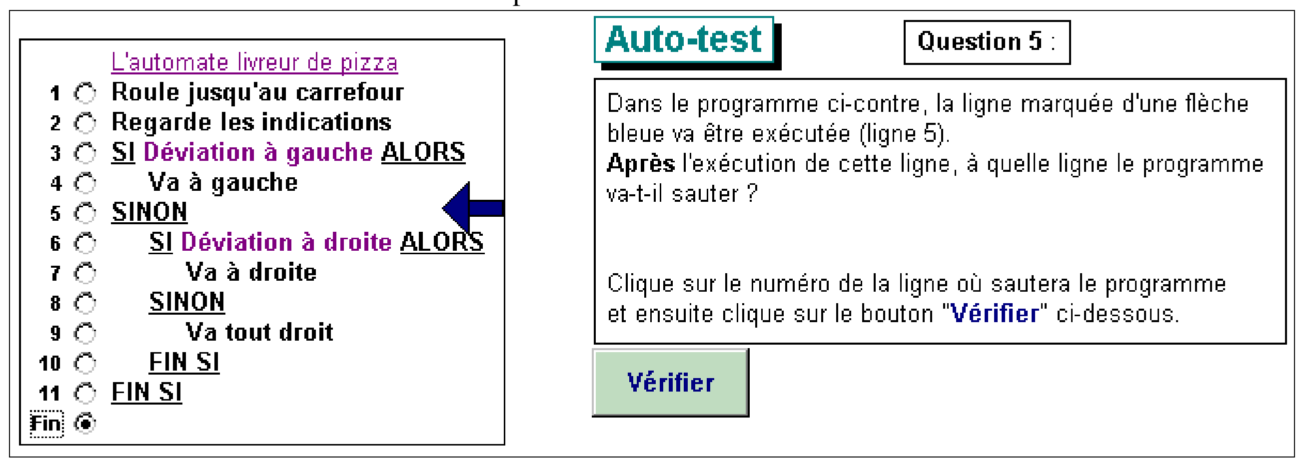


Figure 5

2. Les questions plus globales demandant le résultat de l'exécution de petits programmes.

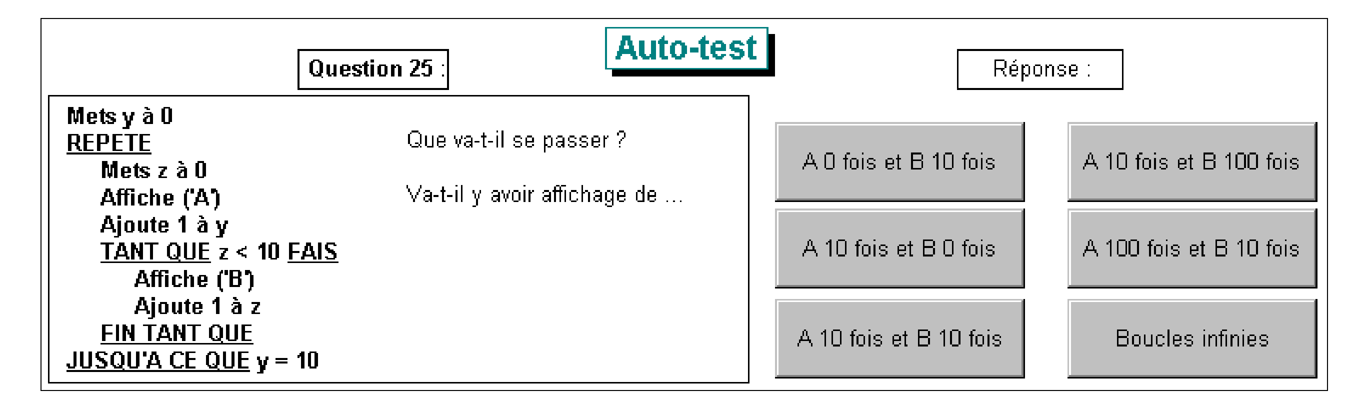


Figure 6

Dans un avenir proche, la trace des réponses fournies par l'apprenant permettra de vérifier dans quelle mesure il a réellement acquis les notions présentées dans ce module.

Au niveau méthodologique, il suffit de laisser l'élève progresser à son rythme et répondre à ses questions éventuelles. La vérification des réponses à l'auto-test en fin de chapitre permettra de vérifier dans quelle mesure il peut continuer le cursus. De toute façon, les performances au module 5 (robotique virtuelle) montreront de façon explicite si les notions des 4 premiers modules sont acquises.

Module 5 : robotique virtuelle

Avec le module 8 (programmation virtuelle en pseudocode), ce chapitre représente le « pierre angulaire » de la méthode. Afin de ne pas trop alourdir ce document, le texte d'une communication faite à Montréal en 1997 se trouve en annexe. Ce texte tente de répondre à la question : « ça marche, mais pourquoi ? ». Il a été réalisé a posteriori sur base du livre de Britt-Mari Barth : « L'apprentissage de l'abstraction ». L'important est de permettre à l'apprenant d'entrer dans l'abstraction sans être coupé de ses ressources, de ce qu'il connaît déjà dans le concret. La virtualité peut être considérée comme ayant simultanément un pied dans le concret (elle peut faire référence à des éléments concrets connus de l'apprenant) et un pied dans l'abstrait (par essence, elle n'existe pas, il est possible de la « disjoncter » du réel), mais ceci est un débat qui pourrait, à lui seul, nécessiter tout un mémoire.

Le chapitre sur l'historique de la méthode explique la manière dont ce module a été développé. Il possède actuellement 17 robots virtuels, mais l'objectif est d'arriver à une cinquantaine. Les robots actuels, regroupés par catégories, sont :

1. Les conditionnelles et leurs imbrications (le but est d'apprendre à utiliser le Sinon et à imbriquer des conditionnelles les unes dans les autres)
   1.1. L'automate lanceur de pièce (une condition pour discriminer deux cas)
   1.2. L'automate livreur de pizza (deux conditions – trois cas)
   1.3. L'automate montreur de cartes (trois conditions – quatre cas)
   1.4. L'automate lanceur de boules (deux conditions – quatre cas)
2. Les opérateurs logiques (manipulation ET, OU, PAS + théorème de de Morgan)
   2.1. L'automate tireur de nombres (utilisation du ET et du OU à bon escient)
   2.2. L'automate frileux (utilisation du ET et du PAS)
   2.3. L'automate faiseur de puzzle (utilisation du ET, du OU et du PAS)
   2.4. L'automate loueur de voiture (application de de Morgan au ET)
   2.5. L'automate sorteur de chien (application de de Morgan au OU)
3. Les répétitives et leurs imbrications (choix entre Répète et Tant que, imbrication de répétitives)
   3.1. L'automate chargeur d'armoires (utilisation du Répète)
   3.2. L'automate transporteur de valises (utilisation du Tant que)
   3.3. L'automate transporteur de repas (2 Répète imbriqués dans un Répète)
4. Imbrication de conditionnelles et répétitives

4.1. L'automate multiple lanceur de pièces (un Si imbriqué dans un Répète)
4.2. L'automate dépouilleur d'enquêtes (deux Si imbriqués l'un dans l'autre et dans un Tant que)
5. Les variables et tours de main (introduction, voir chapitre sur l'organisation de la méthode)
5.1. L'automate gardien de musée (utilisation d'un compteur avec un Répète)
5.2. L'automate remplisseur de bacs de bouteilles (utilisation d'un compteur avec un Tant que)
5.3. L'automate compteur de la valeur d'une liasse de billets (une somme avec un Répète)

Chaque exercice est décomposé en deux pages, une page d'édition et une page d'exécution. La page d'édition contient tous les éléments pour réaliser le programme nécessaire à l'accomplissement de la tâche décrite dans l'énoncé. Le but est de composer, sur le grand tableau « magnétique » gris, le programme en traînant les éléments puisés dans la rubrique « constituants de la marche à suivre » (instructions d'actions élémentaires, conditions, opérateurs logiques et structures de contrôle).

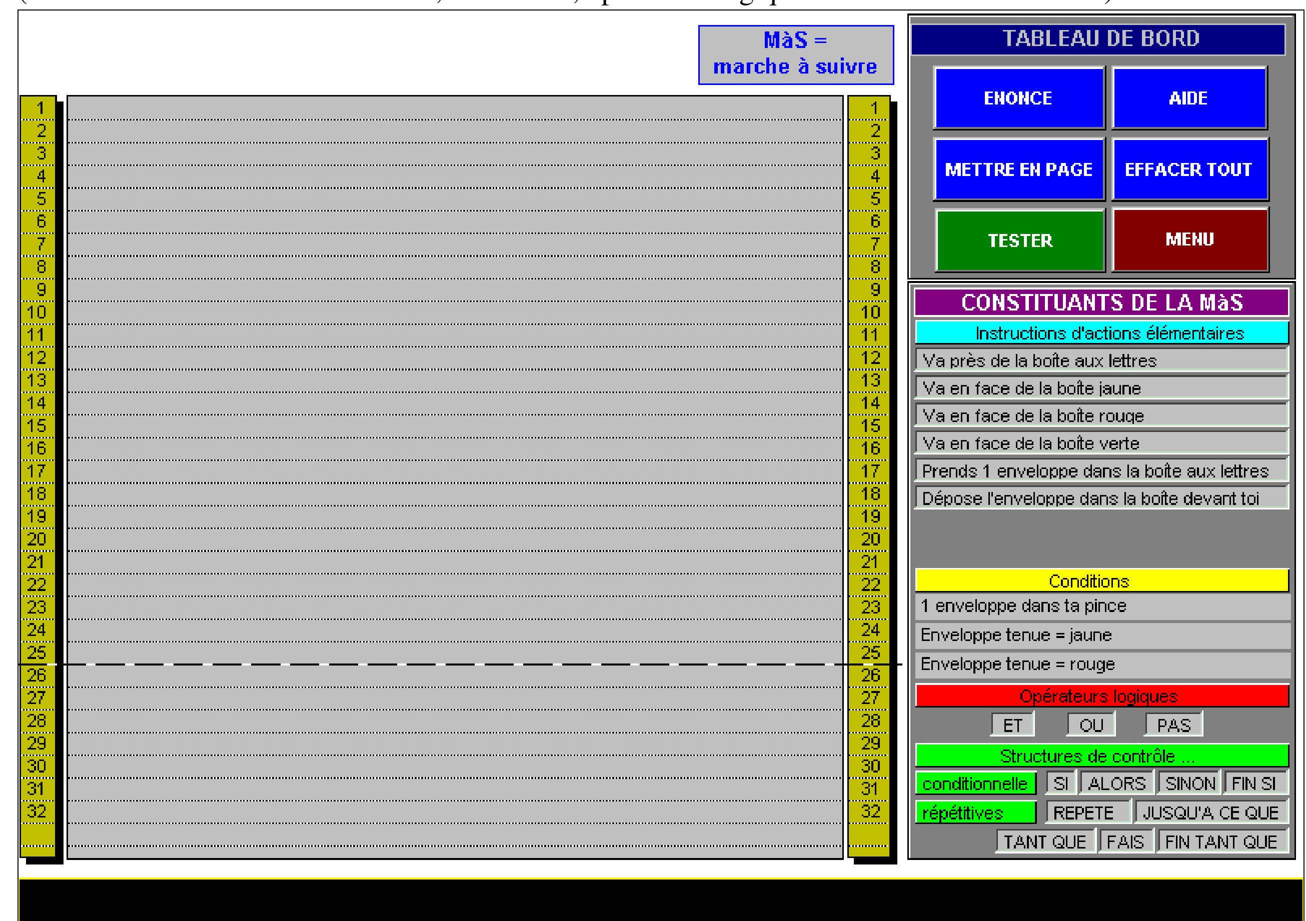


Figure 7

❖ Le tableau de bord contient :
➢ Un bouton permettant d'accéder à **l'énoncé** (voici l'énoncé correspondant à l'exercice de la figure ci-dessus)

Enoncé de l'exercice

Une société effectue une enquête par la poste. Le facteur dépose en vrac dans la boîte aux lettres des enveloppes rouges (pour les fruits), vertes (pour les légumes) et jaunes (pour les fromages). L'automate est chargé de trier ces enveloppes en les déposant dans les boîtes de couleurs correspondantes. Certains jours, il peut n'y avoir aucune enveloppe. Quand l'automate plonge sa pince dans la boîte aux lettres et qu'elle ressort vide, c'est qu'il n'y a plus d'enveloppe dans la boîte aux lettres. Au début, l'automate est près de la boîte aux lettres, et c'est là qu'il doit se trouver en fin de programme.

Le programme sera réussi si les 2 vérifications se concluent avec succès.

Les instructions que l'automate comprend sont :
- *Va près de la boîte aux lettres*, nécessaire pour pouvoir prendre 1 enveloppe
- *Va en face de la boîte jaune*
- *Va en face de la boîte rouge*
- *Va en face de la boîte verte*
- *Prends 1 enveloppe dans la boîte aux lettres*
- *Dépose l'enveloppe dans la boîte qui est devant toi*, quelle que soit sa couleur

Les conditions que l'automate comprend sont :
- *1 enveloppe dans ta pince*, qui teste s'il tient 1 enveloppe dans sa pince ou non
- *Enveloppe tenue = jaune*, qui teste si l'enveloppe dans sa pince est jaune ou non
- *Enveloppe tenue = rouge*, qui teste si l'enveloppe dans sa pince est rouge ou non

Figure 8

- Deux **aides** qui seront bientôt disponibles, une aide technique (comment utiliser les différents éléments du programme) et une aide à la conception (différentes questions permettant à l'apprenant de démarrer sa résolution). Plus tard, le module possédera aussi un petit tutoriel d'introduction afin de faciliter sa prise en main par le débutant.
- Un bouton de **mise en page** qui indente automatiquement les lignes du programme en fonction des structures de contrôle (aide à la lecture et à la vérification du programme)
- Un bouton **d'effacement** du programme pour tout recommencer à zéro
- Un bouton permettant d'accéder à la page de **test**
- Un bouton permettant de revenir au **menu**.

- Les constituants de la marche à suivre contiennent 4 catégories de « sabots »
  - Les **instructions d'actions élémentaires** qui varient en fonction du robot
  - Les **conditions** qui varient également en fonction du robot
  - Les **opérateurs logiques** qui ne varient pas (dans certains exercices, ils ne sont pas disponibles afin de forcer l'utilisation du Sinon)
  - Les **structures de contrôle** qui ne varient pas non plus.

On parle de sabot (comme dans certains jeux de carte) car chacun des champs gris est une réserve d'où on peut tirer autant de copies du contenu qu'on veut, ces copies pouvant être traînées (drag and drop) sur le tableau magnétique gris. Une fois déposées, ces copies peuvent encore être déplacées ou supprimées (voir mode d'emploi). Le cadre noir en bas de l'écran affiche en permanence les indications nécessaires.

Le lecteur intéressé pourra trouver plus de détails dans le mode d'emploi.

Quels sont les intérêts d'une telle interface ?

Comme il a été déjà commenté, il vaut mieux dissocier les difficultés.

Quand on veut programmer, une première difficulté est déjà l'utilisation d'un environnement de programmation dans lequel la dissociation *Édition – Exécution* est rarement évidente[11]. Ce module

[11] Pour du Boulay : « *Une distinction cruciale est à faire entre les différents rôles de l'ordinateur qui sont souvent mélangés par le novice. D'une part se trouve la machine exécutant un des programmes de l'apprenant. Dans ce mode, elle devient le mécanisme décrit par le programme, elle devient l'exécutant. D'autre part, la machine devient*

distingue clairement ces deux phases (il est impossible de modifier le contenu d'un programme en phase d'exécution ni de l'exécuter en phase d'édition) et en permettent l'utilisation très aisée, préparant ainsi le novice à l'utilisation d'un environnement réel de programmation.

L'encodage du programme nécessite la maîtrise du clavier et engendre constamment des erreurs de syntaxe. Se pose aussi le problème de « comment dire les choses »[12]. Dans les exercices, toutes les instructions, conditions,… sont déjà formulées à l'avance et disponibles. L'apprenant peut donc focaliser son attention sur la conception à l'exclusion de tous autres problèmes de frappe ou de syntaxe. Il lui suffit de tirer les éléments sur le tableau magnétique en fonction de ses besoins. La mise en page sous forme d'indentation est automatisée, elle montre son intérêt et, petit à petit, imprègne l'apprenant qui finit par l'adopter.

Une bonne méthode (qui sera mentionnée dans l'aide à la conception) est de suggérer au débutant de programmer par association d'idées (le fil d'Ariane) plutôt qu'en alignant les unes après les autres les instructions du programme de manière séquentielle du début à la fin. Le « tableau magnétique » a été conçu à cet effet. Le concepteur du programme peut très bien placer certaines instructions de la fin d'un programme (comme la condition d'une boucle Répète par exemple) avant de placer ce qui précède. Un simple clic sur le bouton « Mise en page » lui permet d'obtenir son programme en ordre. Par la suite, il lui est encore possible de modifier facilement son programme : la colonne jaune de gauche lui permet de déplacer toute une ligne d'éléments en une seule fois, et la colonne jaune de droite lui permet d'obtenir une ligne vide en repoussant l'ensemble des autres lignes vers le bas. La zone en dessous des pointillés est disponible pour concevoir le programme, mais le programme mis en page ne pourra dépasser cette limite de 25 lignes.

Un problème s'est posé quant à l'utilisation du Fin Si et du Fin Tant Que. La question est ardue et de nombreux courants de pensée ont vu le jour à ce sujet. Il nous a paru pédagogique de placer le novice face à ses responsabilité et de l'obliger à marquer explicitement la fin des instructions faisant partie d'un Si , d'un Sinon ou d'un Tant Que.

Enfin, pour terminer cette partie Édition, le fait de demander la vérification du programme va provoquer, en tâche de fond, une vérification de la syntaxe, c'est-à-dire la manière dont le programme a été construit (et non de la manière dont il a été conçu, la conception étant vérifiée par le test lui-même). Cette vérification est très complète et donne des messages explicites[13]. Tout message signale le type d'erreur trouvé et à quelle ligne elle se situe. L'apprenant peut alors facilement retrouver son erreur et la corriger. L'accès à la page de test n'est autorisé que quand plus aucune erreur de syntaxe ne subsiste dans le programme.

La page de test possède les caractéristiques suivantes :

---

le gestionnaire du programme dans le sens où elle permet de l'éditer, le corriger, le stocker. Dans ce cas, les touches permettent de modifier le contenu du programme et ne correspondent pas aux entrées prévues lors de son fonctionnement (lectures). Il est important de ne pas sous-évaluer ce problème11. Apprendre un langage de programmation ne sera pas seulement apprendre la syntaxe, mais aussi la manière de procéder à l'édition du programme,… »

[12] Certains auteurs signalent un danger important à utiliser le langage naturel (pseudocode) pour apprendre à programmer. du Boulay redoute la dérive que le novice pense que le système soit doté de capacités humaines, Bonar parle carrément de « bug generator » de par la confusion entre la sémantique du langage naturel et celle du langage informatique. Ces deux écueils ne nous sont pas apparus, sans doute par la présentation de primitives déjà rédigées et par leur classement en catégories suivant leurs statuts (un ET utilisé pour former une chaîne d'instructions sera immédiatement repéré par le vérificateur de construction du programme, évitant la prise de mauvaises habitudes).

[13] Pour du Boulay « *les messages d'erreur sont un point crucial de la machine fonctionnelle et forment une fenêtre importante sur l'intérieur de la machine. Un grand soin doit y être apporté afin de s'assurer qu'ils décrivent l'erreur en des termes ou des composants connus du novice.* »

- La partie supérieure représente l'automate et son environnement
- La partie gauche de l'écran est occupée par le tableau magnétique et le programme mis en page
- La partie droite possède tous les boutons et les écrans de contrôle.

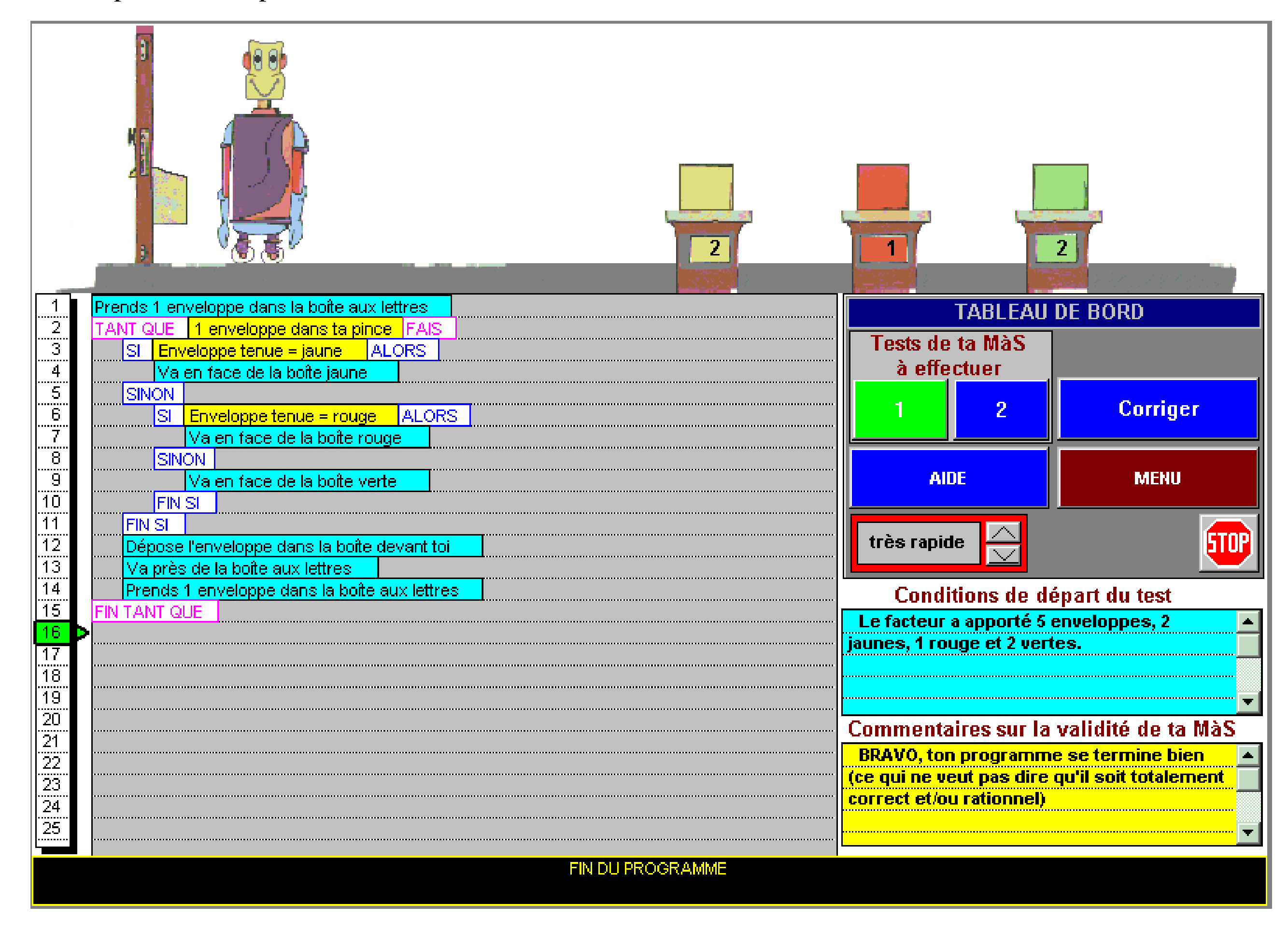


Figure 9

Comme signalé plus haut, le programme doit fonctionner quelles que soient les conditions dans lesquelles l'automate se trouve. Il est souvent difficile, pour le débutant, de concevoir toutes les possibilités dans lesquelles un programme va fonctionner[14]. Le débutant va concevoir son programme généralement pour un sous-ensemble des conditions d'entrée possibles et ensuite le tester sur ce même sous-ensemble, ce qui lui fera dire que son programme est correct sans qu'il se rende compte que sa validation est incomplète. Pour éviter cela, nous avons défini, pour chaque automate, les différentes conditions de fonctionnement dans lesquelles il pouvait démarrer. Chacune de ces possibilité a donné lieu à un bouton de test numéroté. Certains robots possèdent 2 boutons de test, d'autres jusqu'à 6 en fonction des circonstances. Quand un test est terminé, le bouton bleu prend la couleur verte si l'exécution n'a provoqué aucun problème, il devient rouge en cas d'erreur (plantage, résultats mauvais ou incomplets). L'exercice est quasiment réussi quand tous les boutons d'un même exercice sont verts. L'avis du professeur doit alors être sollicité. La vérification automatique de la rationalité d'un programme est à l'étude (partant de l'hypothèse que le nombre de conditions testées en cours d'exécution est un bon indicateur) mais l'avis du professeur est préférable car il faut éviter à tout prix que le débutant prenne de mauvaises habitudes de programmation. De plus, certains énoncés permettent plusieurs types de solutions sans qu'il y en ait nécessairement une « meilleure » que les autres. Il est même parfois très intéressant de pouvoir discuter des avantages et

[14] Spohrer signale, au niveau des erreurs de conception, les « *cas inattendus : le novice écrit des programmes qui fonctionnent dans certains cas mais pas dans tous* ».

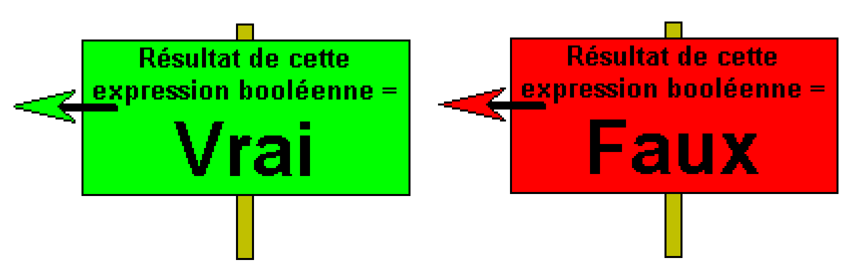
inconvénients des différentes solutions. Ce type d'évaluation doit donc rester, en fin de compte, l'apanage de l'enseignant.

Deux types d'exécutions sont possibles : l'exécution automatique, au cours de laquelle tout le programme est exécuté du début à la fin, et l'exécution pas à pas, où l'utilisateur doit cliquer sur un bouton pour obtenir l'exécution de l'instruction suivante. Ce mode d'exécution est particulièrement intéressant soit pour débugger un programme ou en phase de tutorat pour demander à l'apprenant de prédire ce qui va se passer à la ligne suivante, excellente façon de vérifier s'il maîtrise le fonctionnement des structures de contrôle.

Les autres constituants du tableau de bord sont :

- Le bouton **Corriger** permet de retourner à la page d'édition
- Le bouton **d'Aide** permettra d'avoir des détails sur les constituants de la page
- Le bouton **Menu** permet de choisir un nouvel exercice
- L'indicateur de **vitesse** d'exécution qui peut être réglé pour ralentir les mouvements des automates
- Le bouton **Stop** permet d'arrêter un programme qui tourne sans fin en exécution automatique
- Le cadre bleu donne les **conditions de départ** de l'automate
- Le cadre jaune donne des commentaires en **fin d'exécution** du programme, notamment en cas d'erreur de façon à aider l'utilisateur à découvrir le problème.

Durant l'exécution du programme, un pointeur vert signale la ligne en cours de traitement. S'il s'agit d'une condition, une panneau indicateur apparaît afin de signaler si le test a donné un résultat vrai (panneau vert) ou faux (panneau rouge).



Les « flow-charts » telles que le GNS sont intéressantes pour vérifier le flux d'un programme, ce qui est à la fois éducatif au niveau du fonctionnement des structures de contrôle, mais surtout une aide importante pour découvrir les erreurs de programmation. Apprendre à concevoir ces flow-charts est une difficulté supplémentaire pour le débutant pour un intérêt qui ne sera que momentané (ces représentation sont généralement abandonnées assez rapidement, elles deviennent fastidieuses quand les programmes deviennent relativement complexes). L'idée est d'avoir « le beurre et l'argent du beurre » en traduisant automatiquement le programme réalisé en GNS. L'apprenant, sans devoir maîtriser des savoir-faire supplémentaires, aurait la possibilité de tester son programme soit sous forme séquentielle (comme c'est déjà le cas pour le moment) soit sous forme de GNS qui montrerait mieux le flux d'exécution du programme. Cette implémentation est à l'étude.

Toutefois, dans l'exécution séquentielle, des indicateurs permettent de repérer les différents éléments des structures de contrôle actives. Dans l'exemple ci-dessous, le curseur vert indique la ligne en cours de traitement. Les indicateurs mauves indiquent les positions des éléments constitutifs du Tant que. Les indicateurs bleus indiquent les positions des éléments constitutifs des Si, les indicateurs bleus grisés correspondant au Si en cours d'exécution. Un bon exercice serait de montrer cette portion de programme à un apprenant et lui demander, sachant que la ligne 9 est en cours d'exécution, quels étaient les résultats des tests des lignes 2, 3 et 6.

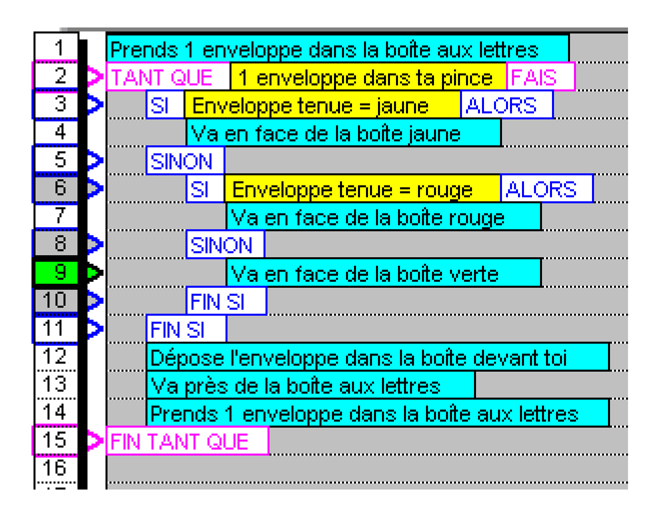


Figure 10

Comme il a été dit plus haut, une réflexion est en cours pour voir comment, dès les premiers programmes, une démarche procédurale pourrait être mise en œuvre. Cette démarche nous paraît plus intéressante que la démarche actuelle car elle favoriserait la démarche descendante et structurée qui sera indispensable pour les programmes plus complexes ou plus volumineux. Cela éviterait également de devoir introduire ce nouveau concept plus tard (cf. les sauts dans l'abstrait).

Module 6 : les spécificités du robot-ordinateur

Au niveau des spécificités du robot-ordinateur, il va être essentiellement question du transfert, du stockage et de la transformation d'information. Il fallait trouver une analogie adéquate afin de présenter un système homogène au débutant. Après mûre réflexion, l'utilisation d'autocollants nous a paru la plus judicieuse. Attention, il s'agit d'autocollants opaques et munis de colle forte. En d'autres termes, quand un autocollant est collé sur un autre, l'ancien autocollant est devenu invisible et inaccessible, et le nouvel autocollant ne peut plus être détaché. Voici la saga des autocollants.

Comme signalé plus haut, l'ordinateur est considéré comme un robot ayant des spécificités propres. À l'entrée du chapitre, une métaphore anthropomorphique est présentée.

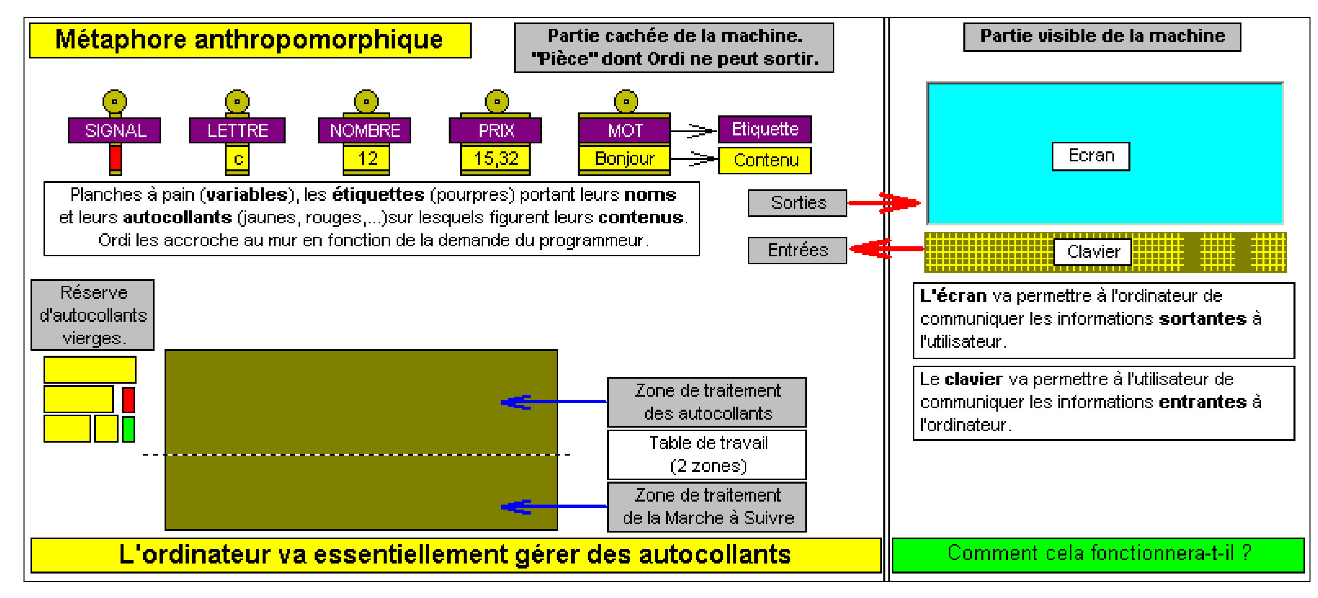


Figure 11

L'opération de lecture au clavier est souvent présentée comme banale, mais représente des difficultés pour certains débutants. Ils ne comprennent pas toujours que la variable associée à l'instruction de lecture

est affectée (du Boulay). Un résumé de ce qui se passe est présenté à l'apprenant (suivez la flèche bleue et les numéros des étiquettes grises). Remarque : une étiquette 4bis sera ajoutée signalant qu'une copie de ce que l'utilisateur tape au clavier apparaît à l'écran. Sans avoir l'air d'y toucher, le concept d'affectation d'une variable est abordé.

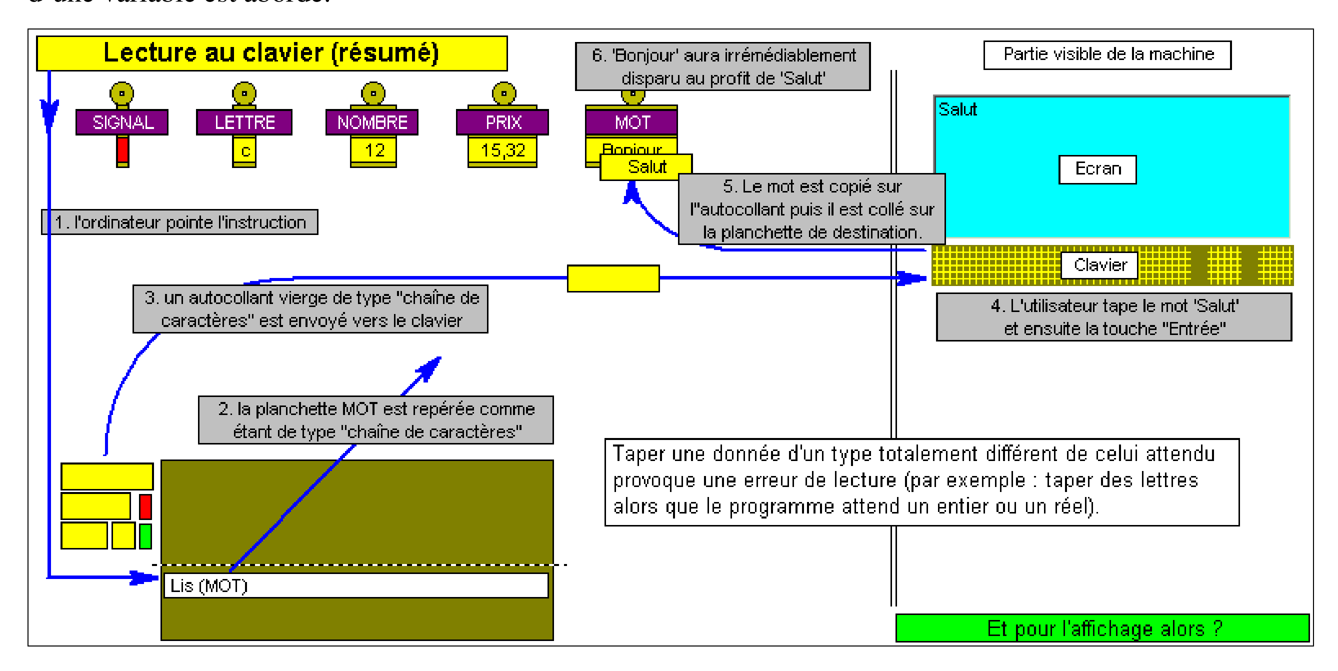


Figure 12

Dans le cas présenté, l'affichage à l'écran va mettre en œuvre le concept de lecture d'une variable. Le principe adopté est que, lorsque l'ordinateur trouve le nom d'une variable dans la ligne qu'il est en train de traiter, il va chercher une copie du contenu de cette variable sur un autocollant et vient le coller sur le nom de la variable stipulé dans la ligne.

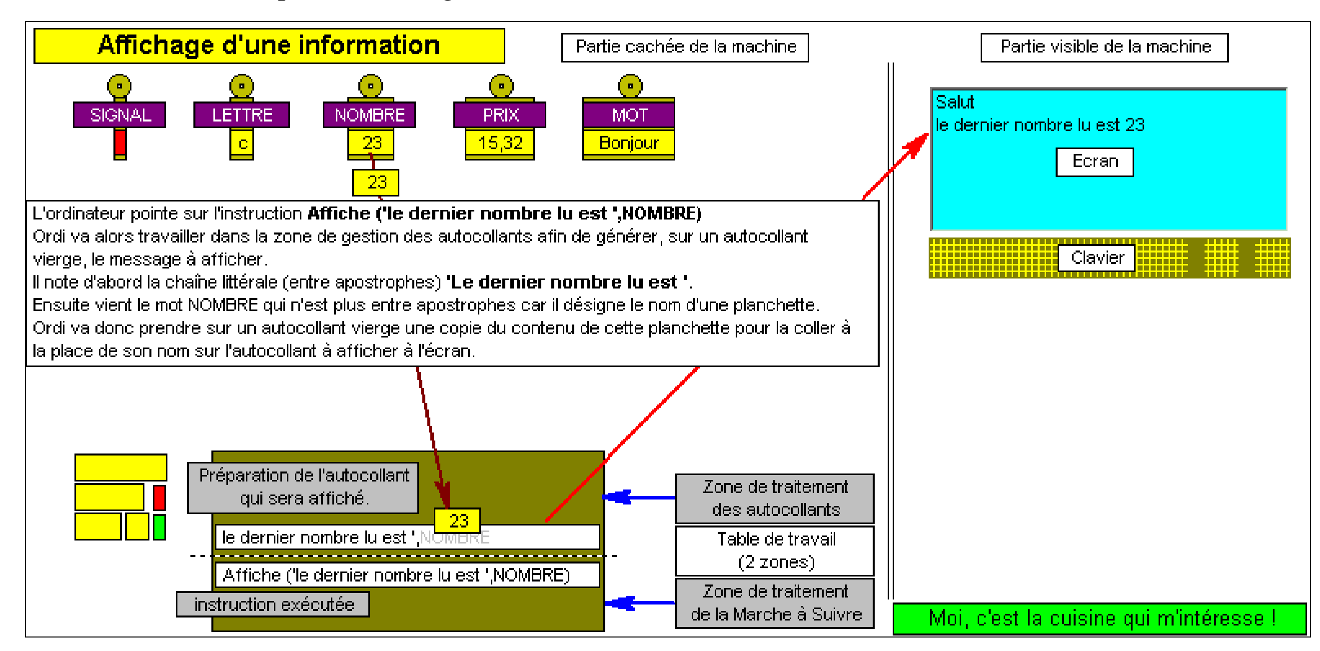


Figure 13

La qualité d'une analogie est d'être complète, correcte, ne pas être source d'erreurs et ne pas déboucher sur une impasse par la suite. Une bonne analogie pour illustrer le concept de variable doit répondre aux critères suivants[15] :

1. Une variable sans étiquette ne peut être atteinte
2. La notion de type doit être prise en compte
3. Une variable ne peut être lue avant d'y avoir inscrit « quelque chose »
4. La variable ne contient que la dernière information enregistrée[16]
5. La variable contiendra une constante et non une expression
6. La lecture d'une variable ne vide pas son contenu
7. L'analogie doit pouvoir fonctionner au moment de l'utilisation des paramètres.

Si certaines de ces affirmations sont évidentes (la première par exemple), d'autres le sont moins. L'analogie des planchettes à autocollants nous paraît répondre à tous ces critères, y compris le dernier : une planchette peut être enlevée de son clou, y être attachée par l'intermédiaire d'un élastique et envoyée dans une procédure. En fin de procédure, elle viendra reprendre sa place originale.

Voici une illustration de l'analogie des planchettes à autocollants (la largeur symbolise le type).

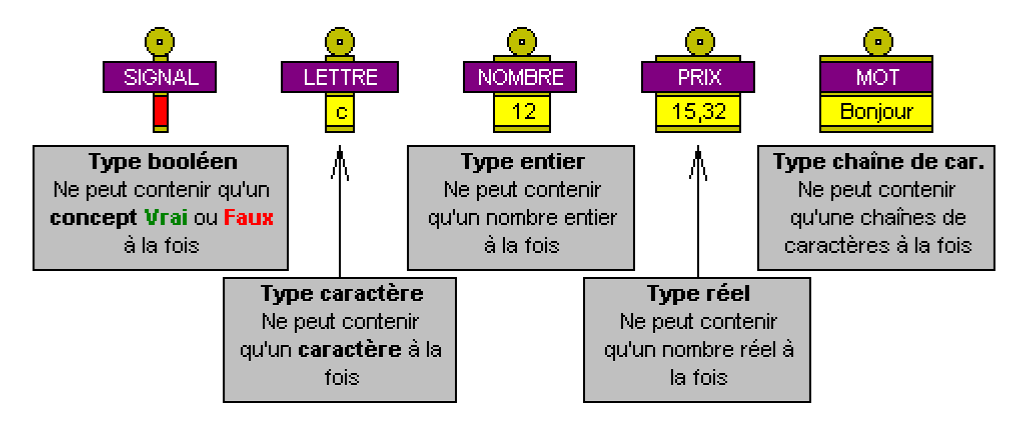


Figure 14

Voici comment est présentée l'affectation d'une variable avec une valeur résultant d'une expression mathématique faisant elle-même intervenir une variable.

---

[15] du Boulay signale un florilège d'erreurs au niveau des variables : « *le fait que le contenu d'une variable puisse disparaître ou qu'on puisse se souvenir d'un ancien contenu de variable; une autre confusion est de croire que quand on assigne une variable avec l'expression 7 + 4, c'est cette expression littérale qui s'inscrit dans la variable plutôt que la valeur 11; certains croient que quand on assigne une variable dans une autre, la variable de départ se vide.* »

[16] Putnam signale que « *l'erreur la plus fréquente est de croire qu'une variable peut contenir plus d'une valeur. Une autre erreur* (quand on demande à l'élève d'expliquer à quoi sert un programme) *est d'utiliser toujours la même valeur initiale de la variable sans procéder à sa mise à jour.* »

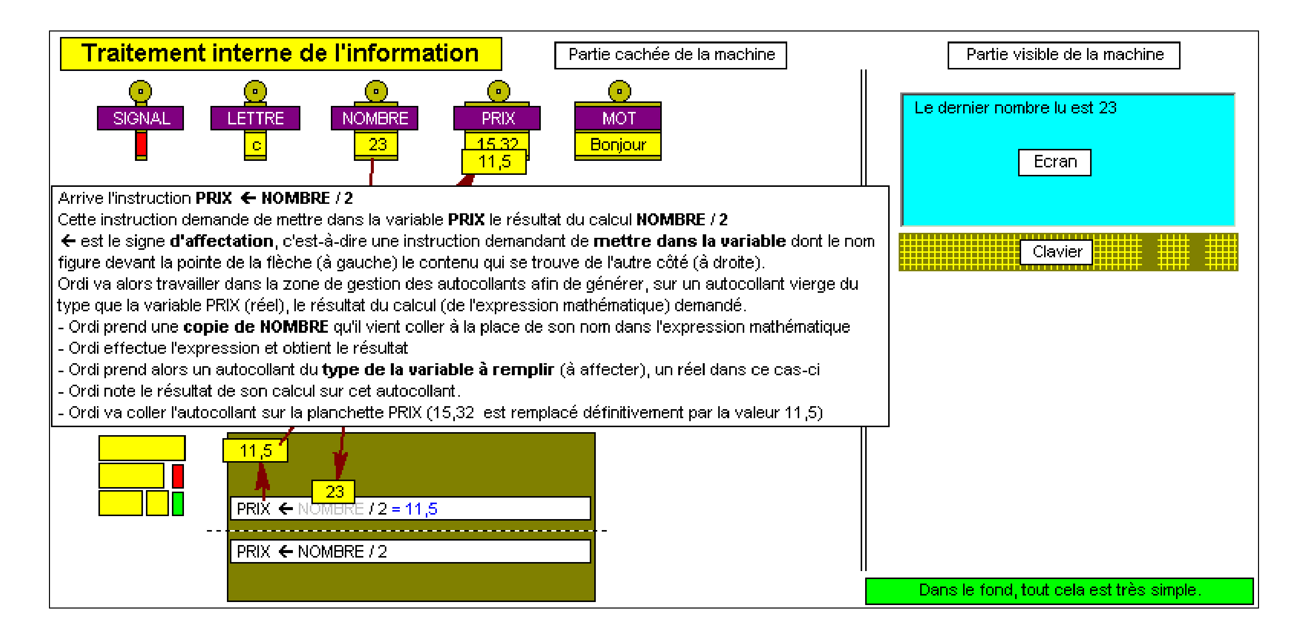


Figure 15

Comme pour les autres module, un auto-test permet à l'apprenant de vérifier ses acquis. Les questions portent d'abord sur les contenus des variables après plusieurs affectations, ensuite sur la compatibilité des types (le débutant ne se rend pas toujours compte qu'une expression peut changer le type des données obtenues). Voici deux exemples de questions.

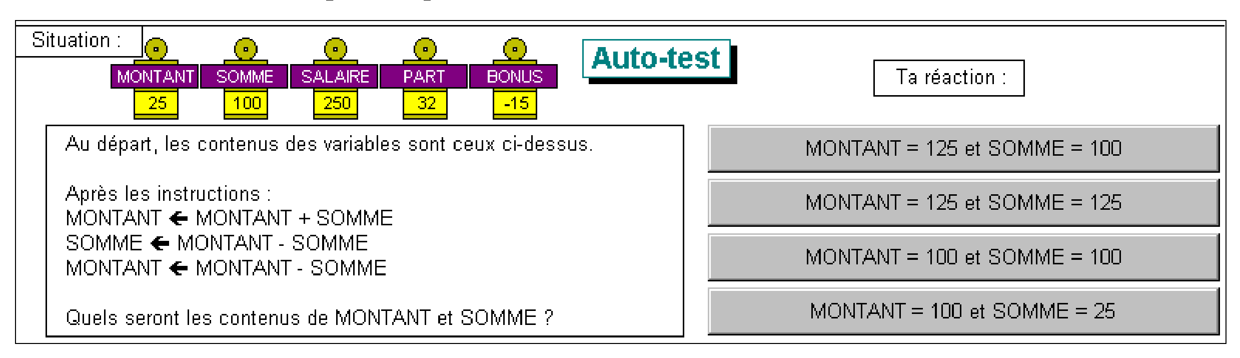


Figure 16

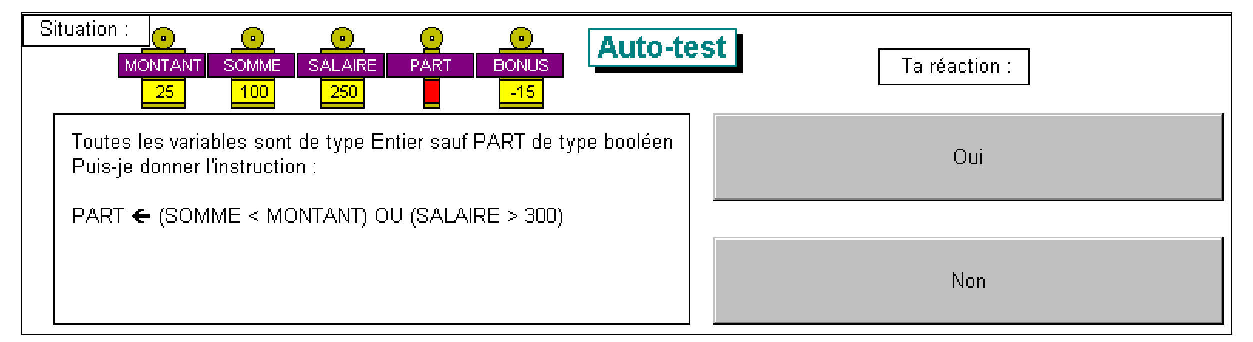


Figure 17

Remarque : bien que la lecture des programmes soit facilitée par l'utilisation de noms de variables « qui veulent dire quelque chose », cela entraîne certains débutants à penser qu'il y a un lien entre le nom de la variable et son contenu (du Boulay, Putnam). Dans une version ultérieure, afin d'éviter ce genre de confusion, certaines variables seront parfois appelées par des noms « qui ne veulent rien dire ».

Module 7 : les tours de main

Images pour programmer met en exergue un certain nombre de petits programmes qui « aident la vie » lors de la conception de programmes plus importants ; ils sont appelés « tours de main ». Ceci rejoint le conseil de du Boulay suggérant l'acquisition par le débutant de structures standards à utiliser pour accomplir certaines tâches de base.

Ce module présente actuellement 6 tours de main (tous empruntés à « Images pour programmer ») :

1. Le compteur
2. La variable signal
3. La somme
4. Le plus petit et le plus grand
5. Le dernier et le précédent
6. Les positions du premier et du dernier

Un septième tour de main sera bientôt ajouté afin de faciliter la vie des apprenants auquel il est systématiquement demandé des programmes robustes, ce sera le tour de main de « bonne lecture » qui évitera la lecture de données indésirables (un 7 comme valeur à obtenir pour un dé ou un –5 pour le nombre de candidats aux élections par exemple).

Chaque tour de main est développé en détail, mais un « raccourci » permet aux plus rapides de sauter directement au tour de main fini afin d'en examiner la structure. S'ils ne la comprennent pas, un retour en arrière est toujours possible.

Le programme est alors exécuté en utilisant la métaphore anthropomorphique développée au module précédent.

Ce chapitre n'est pas muni d'un auto-test, le module suivant révélera sans problème les lacunes des apprenants.

Module 8 : programmation virtuelle en pseudocode

Le module 8 a une interface très semblable à celle du module 5 (robotique virtuelle) mis à part la présence de variables et les constituants de la marche à suivre sont adaptés à ce type de robot.

Au-dessus du tableau magnétique se trouve 8 variables configurables par le programmeur, il peut à tout moment y indiquer un nom et un menu déroulant lui permet de choisir entre les 5 types standard du Pascal (booléen, caractère, entier, réel, chaîne de caractères). Une variable ainsi déclarée peut à tout moment être « traînée » (drag and drop) sur le tableau magnétique.

Les instructions d'actions élémentaires sont au nombre de 4 :

1. **Affiche+nl**, pour afficher à l'écran et faire ensuite passer le curseur au début de la ligne suivante
2. **Affiche**, pour afficher à l'écran, mais le curseur reste derrière le dernier caractère affiché
3. **Lis et colle sur**, pour lire une donnée au clavier et affecter la variable dont le nom suit (va coller l'autocollant sur la planchette portant ce nom)
4. **←**, pour affecter une variable[17].

Viennent ensuite les constantes de types numérique et caractère(s) (que le programmeur peut compléter lui-même et traîner ensuite sur la tableau magnétique), ainsi que les constantes Vrai et Faux. Le contenu des constantes caractère(s) apparaissent en caractères normaux dans leurs cadres jaunes, et les constantes numériques en gras. Toutefois, cette différence est trop ténue et, dans une version ultérieure, les

[17] Ceci évite les erreurs signalées par une foule d'auteurs, soit la confusion avec le signe =, soit l'affectation « à l'envers » (de la gauche vers la droit plutôt que de la droite vers la gauche). La disposition du = dans une catégorie différente du ← permet d'attirer l'attention du débutant sur les statuts de ces deux signes.

constantes caractère(s) apparaîtront entre apostrophes, comme c'est le cas en Pascal, afin d'attirer correctement l'attention sur leur statut.

Les opérateurs mathématiques sont :

- + , - , * , / les 4 opérations de base
- DIV fonction donnant le quotient de la division entière
- MOD fonction donnant le reste de la division entière
- Hasard fonction donnant un nombre tiré au hasard entre les deux bornes indiquées comprises

Les divers contiennent : (,) , [,] (les crochets n'ont pas encore d'utilité)

Les opérateurs booléens sont ET , OU , PAS

Les comparateurs sont = , < , > , <= , >= , <>

Viennent enfin les différentes structures de contrôles. Tout cela donne l'écran suivant.

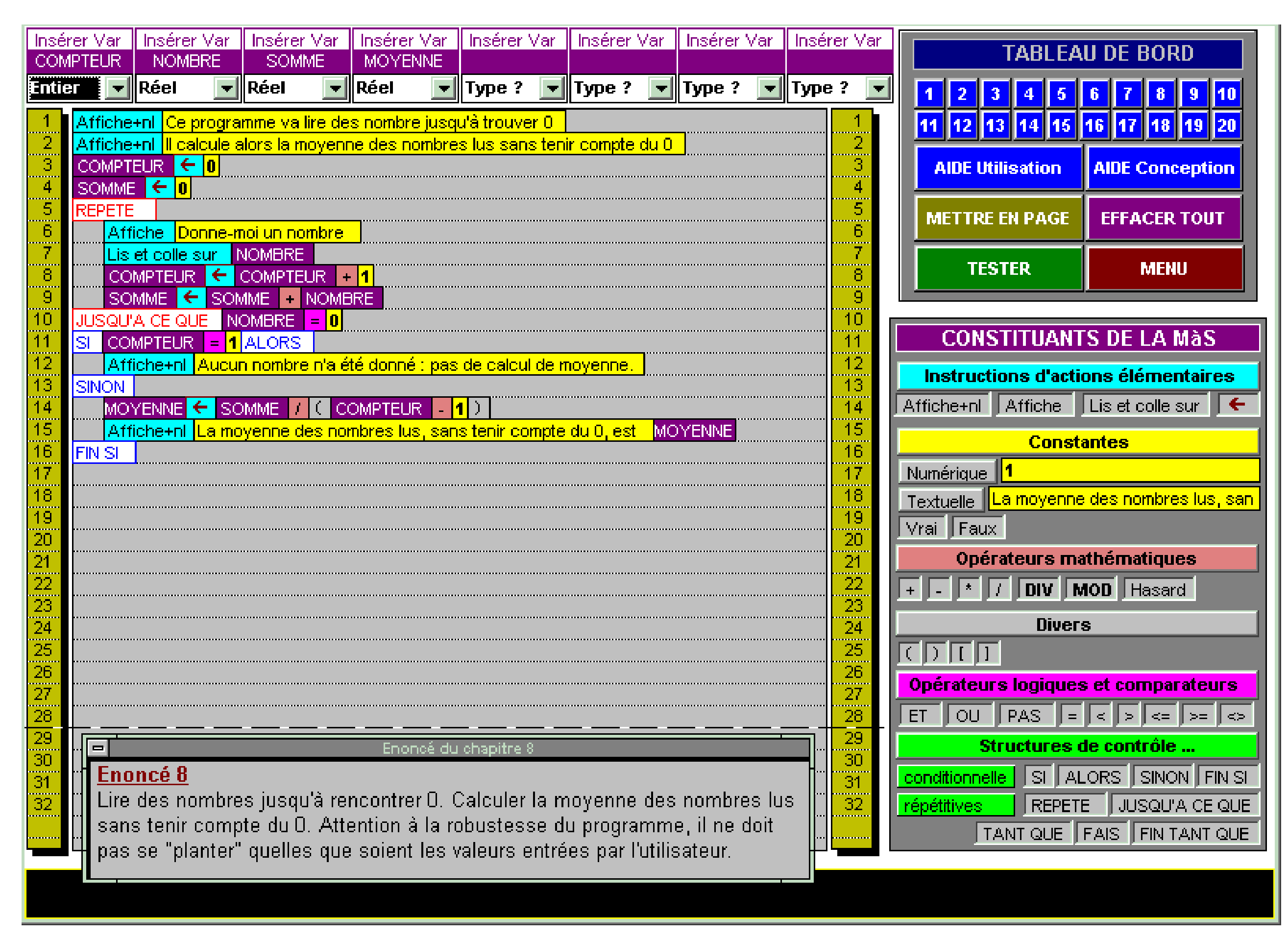


Figure 18

L'avantage de ce genre de regroupements est que l'apprenant, dès le départ, a l'attention attirée sur les différents statuts des éléments qu'il utilise pour concevoir son programme. Ceci est essentiel lorsqu'il devra rechercher des éléments du langage dans un livre par exemple ou comprendre la littérature spécialisée. Dans les versions ultérieures, il sera veillé à ce que la rigueur des appellations soit améliorée.

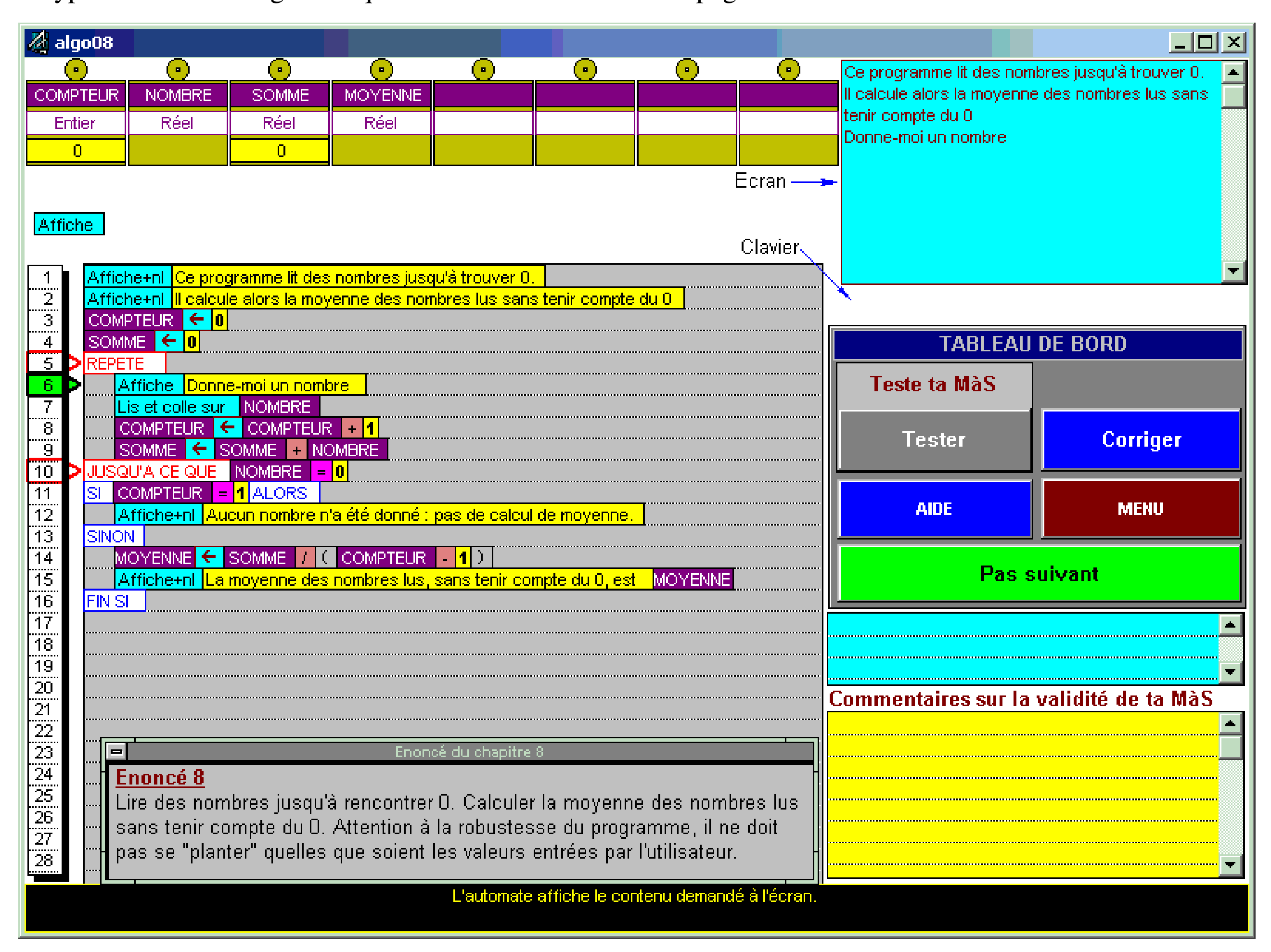
L'énoncé est obtenu en cliquant sur les boutons numérotés de 1 à 20. L'énoncé peut être déplacé et redimensionné. Il peut de plus rester affiché en permanence, tant sur la page d'édition que sur la page d'exécution.

L'utilisateur construit son programme de la même façon que dans le module 5.

Les mêmes caractéristiques sont à remarquer au niveau de la facilité d'utilisation, du manque d'erreur de syntaxe, de visualisation des statuts des éléments constitutifs.

Précisons qu'une variable doit nécessairement être nommée ET typée avant de pouvoir être utilisée, ce qui est également très éducatif. En cas d'erreur de type, il suffit de le modifier dans la page d'édition au niveau du menu déroulant et demander une nouvelle exécution. Le type sera automatiquement mis à jour.

Une vérification de la syntaxe est réalisée au moment du passage à la page de test avec message signalant le type d'erreur et la ligne à laquelle elle se trouve. Voici la page d'exécution.



Au niveau de la page d'exécution, il y a une nouveauté : chaque ligne traitée est amenée sur la « table de gestion des autocollants ». les manipulations nécessaires y sont réalisées afin d'exécuter l'instruction. Quelle que soit la manipulation (affichage, lecture, affectation, test de condition, résolution d'une expression mathématique,…) l'entièreté du traitement est animé. Le transfert et le collage des autocollants est montré explicitement de façon à favoriser la création d'images mentales et l'ancrage de la représentation du fonctionnement de la machine fonctionnelle. Pour les expressions booléennes, les autocollants changent de couleurs au fur et à mesure des résultats du traitement suivant que le résultat est vrai (vert) ou faux (rouge).

Chaque variable est présentée avec son nom, son type et son contenu. Une variable non encore affectée ne présente aucun autocollant de contenu; sa lecture provoquera une erreur avec message explicite[18]. Au

[18] Selon du Boulay : « *Une difficulté majeure est l'initialisation des variables. Il est typique d'oublier d'initialiser la variable "Somme" car l'analogie est faite avec une boîte, et si personne n'a utilisé cette boîte avant, elle est forcément vide, et ce vide représente d'office le zéro. Certains systèmes exacerbent cette mauvaise compréhension en assignant d'office des valeurs par défaut aux variables non assignées.* ». Ceci suppose que l'apprenant vérifie le

moment des calculs et des affectations, les erreurs de types sont signalées de façon précise (messages d'erreur). Voici comment se présente une erreur d'affectation, toutes les variables étant cette fois déclarées de type entier.

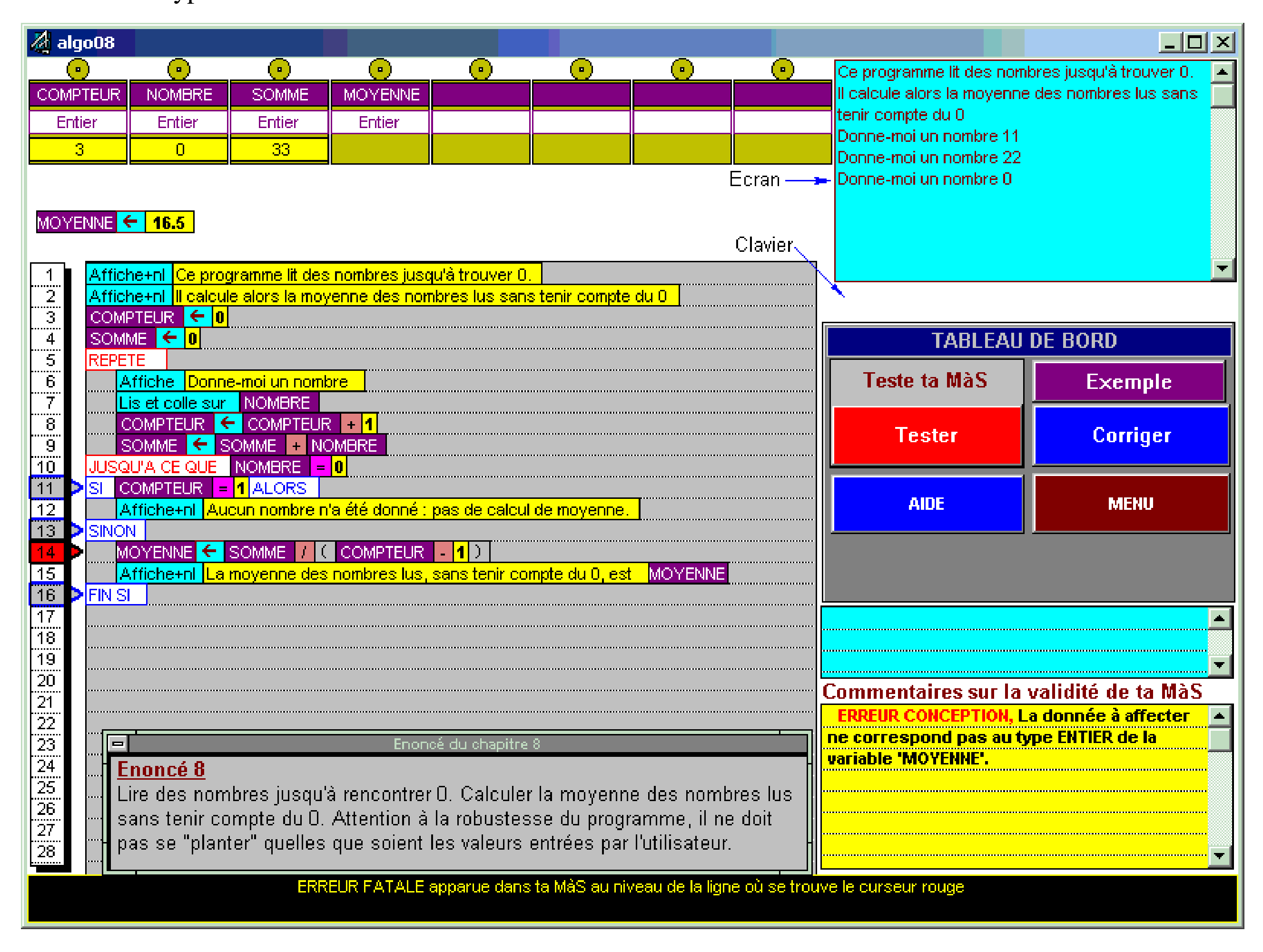


Figure 19

Une seule page d'édition est actuellement prévue pour les 20 programmes plus tous ceux que peut taper l'apprenant, programmes de sa propre conception ou dont l'énoncé a été conçu par l'enseignant. Dans une prochaine version, il est envisagé, comme pour les robots, de créer un exercice par programme, ce qui permettrait, dans une certaine mesure, des exécutions automatiques avec des listes de données sensibles (valeurs pouvant provoquer une erreur, ce qui serait le cas dans le programme présenté si, comme cela arrive souvent, l'apprenant n'avait pas imaginé que l'utilisateur puisse donner directement la valeur 0 (risque de division par 0)). Même dans cette architecture, une page d'édition resterait libre afin de pouvoir taper des programmes répondant à des énoncés extérieurs au module.

Le programme est exécuté pas à pas, mais il est possible d'envisager une exécution automatique.

Parmi son relevé des difficultés des débutants, du Boulay signale « *qu'il est rarement clair pour un novice que l'affichage à l'écran représente un enregistrement des interactions précédentes entre l'utilisateur et la machine…* ». Or, normalement, lorsque l'écran est complet, les nouveaux affichages s'ajoutent en bas de l'écran, provoquant au fur et à mesure la disparition des affichages du haut. Ce phénomène est néfaste car, quand le professeur est appelé en fin de programme pour valider celui-ci ou pour expliquer les raisons de son dysfonctionnement, certaines informations risquent d'avoir disparu. Il

---

calcul effectué par l'ordinateur pour remarquer l'erreur, ce qui est rarement le cas. Nous avons opté pour une solution plus simple : montrer explicitement que la variable est toujours vide et générer un message d'erreur.

lui est alors impossible de prendre connaissance de l'historique de ces échanges. Pour éviter cet inconvénient, le champ servant d'écran a été muni d'un « ascenseur » afin que le professeur puisse récupérer tout l'historique des échanges et, éventuellement, expliquer à l'apprenant les causes de l'erreur.

Les apprenants sont très motivés par ce module et fournissent des programmes de qualité.

La plus grande difficulté, au moment du passage à la syntaxe Pascal, est la gestion des Begin et End dans l'utilisation du IF et du While. Après quelques erreurs « incompréhensibles » et une étude correcte de leurs feuilles de syntaxe, ce problème s'estompe.

Module 9 : la syntaxe Pascal

Un résumé de la syntaxe Pascal est présenté. Module à parfaire avec des exemples, des cas concrets.

Module 10 : exercices de programmation en Pascal (énoncés)

Neuf énoncés sont proposés. À étoffer.

Module 11 : les variables indicées

Module en cours d'amélioration

Module 13 : les procédures paramétrées

Module en cours d'amélioration

Apport du DES-TEF

Les apports du DES-TEF ont été de trois ordres :

◊ Les apports théoriques par l'ensemble des modules qui m'ont permis de d'augmenter mes connaissances en pédagogie, évaluation, systèmes et dispositifs de formation et d'éducation, conception et réalisation multimédia, dispositifs de formation à distance, contrôle de qualité.

◊ Les apports pratiques au travers un grand nombre de travaux seul ou en groupe. La confrontation avec le réel permet de prendre connaissance de vrais problèmes et d'apprendre à les résoudre.

◊ Les apports des équipes qui ont choisi mon logiciel comme base de travail, soit pour en analyser la conception, soit pour en évaluer la qualité, soit pour s'intéresser au phénomène d'innovation lui-même.

Tous ces apports sont vraiment importants et profonds, je me rends compte qu'il me faudra encore du temps pour en tirer les pleins profits par le truchement des nombreux documents et références dont j'ai fait « le plein ».

Je suis maintenant paré pour terminer mon didacticiel dans les meilleures conditions afin qu'il puisse aider les jeunes à prendre possession de leurs pleines possibilités intellectuelles et éventuellement pratiquer un métier que je leur aurai fait découvrir.

Fiche technique

Ce didacticiel est construit sur base du progiciel ToolBook II version 6.0

Sa taille totale est actuellement inférieure à 10 Mo.

Il tourne sur toutes les versions de Windows et toutes les machines à partir des 486, la taille de la mémoire est indifférente.

Ce logiciel tourne sur poste seul (possibilité de sauver les travaux des modules 5 et 8) et sur réseaux Linux, Novell, Windows, etc. En cas d'installation en réseau, les sauvegardes des modules 5 et 8 peuvent également s'effectuer en donnant comme destination un répertoire ouvert en écriture.

Conclusions

En guise de conclusion, nous pourrions nous demander pourquoi il est utile de se démener pour améliorer les méthodes d'apprentissage de la programmation à une époque où elle a quasiment disparu de nos écoles secondaires.

Je vois trois réponses à cela :

◊ Premièrement parce que j'ai constaté que ce cours est bénéfique pour les élèves. Ils améliorent leurs capacités de logique et d'abstraction, même si des expériences n'ont pas encore prouvé ce fait de façon indubitable.

◊ Deuxièmement, on pourrait se demander pourquoi les cours de programmation ont disparu des écoles. J'attribue cela à deux causes : d'une part la difficulté des méthodes employées qui rendent ce cours ardu pour des débutants adolescents, d'autre part, la formation des enseignants n'était et n'est pas à la hauteur.

◊ Troisièmement, un retour timide de la programmation voit le jour dans les écoles secondaires belges via l'option « Sciences et Informatique ». Toutefois, ces cours semblent reprendre dans les mêmes conditions qu'ils ont commencé il y a 10 ou 20 ans : avec des enseignants mal formés utilisant des méthodes incertaines.

Au vu de ces trois arguments, il est tout à fait pertinent de développer ce genre de méthode pour la programmation puisse retrouver une place de choix dans le secondaire et aider les élèves à développer mieux encore leurs facultés intellectuelles.

En tout cas, en ce qui me concerne, j'ai constaté une réelle amélioration des capacités logiques et d'abstraction chez nombre de mes élèves, et, si c'était à refaire, je le referais.

Remerciements

Tous mes remerciements vont d'abord à Charles Duchâteau pour toute la passion, la fougue, l'honnêteté intellectuelle et le travail dont il fait preuve au service des autres. J'ai toujours pu compter sur son aide. Merci aussi à Étienne qui à l'occasion me fait part de ses conseils et n'hésite pas à faire connaître mon logiciel.

Tous mes remerciements pour Brigitte, Bernadette, Marguerite, Sébastien, et tous les intervenants du DES qui n'hésitent pas à investir temps, labeur, intelligence et gentillesse pour faire du DES-TEF la véritable réussite qu'il est.

Grand merci à Patrick et Pierre pour leur travail important d'examen et d'amélioration du logiciel.

Merci à Adèle, Pierre (encore) et Amaury qui se sont penchés sur mon travail selon l'angle particulier de l'innovation.

Merci à Michel d'avoir « épluché » mon logiciel dans le moindre détail.

Merci aussi à tous les participants au DES de cette année et de l'année passée. Nous formions, formons et formerons une bonne équipe.

Merci à mes élèves qui collaborent à l'amélioration de la méthode et à mes proches qui me supportent

Merci à Jean-Baptiste Sonnet qui a dessiné les robots (les beaux).

Bibliographie

Barr A., Beard M., Atkinson R. C., *The computer as a tutorial laboratory : the Stanford BIP project* International journal of man-machine studies (1976) 8, pp. 567 à 596

Barth B. M., *L'apprentissage de l'abstraction*, Paris, Retz

Cunniff N., Taylor R. P., Black J. B., *Does programming language affect the type of conceptual bugs in beginners's programs ? A comparison of FPL and Pascal* in Studying the Novice Programmer, University of Michigan 1989 PP 419 à 429

du Boulay B., *Some difficulties of learning to program* in Studying the Novice Programmer, University of Michigan 1989 PP 283 à 299

du Boulay B., O'Shea T., Monk J. B., *The black box inside the glass box* in Studying the Novice Programmer, University of Michigan 1989 PP 431 à 446

Duchâteau Ch., *Faut-il enseigner l'informatique à ses utilisateurs ?* Communication au 4° colloque francophone sur la didactique de l'informatique, Québec, Formation-recherche en éducation n° 5.38, Publications du CeFIS, Namur (B), Facultés Notre-Dame de la Paix

Duchâteau Ch., *Images pour programmer*, CeFIS, DET, FUNDP, Namur. Consultable sur Internet à l'adresse http://www.det.fundp.ac.be/cefis/publications/doconline.html

Duchâteau Ch., *Robotique-Informatique mêmes ébats, mêmes débats, mêmes combats ?* in Denis B. et Baron G.-L., Regards sur la robotique pédagogique, Actes du quatrième colloque international sur la robotique pédagogique, Liège (B), Université de Liège, pp 10 à 33

Duchâteau Ch., *Socrate au pays des ordinateurs,* revue de l'EPI n° 74, juin 1994 pp 159 à 177

Haugulstaine-Charlier B., *Images pour apprendre à programmer…* EPI Colloque francophone sur la didactique de l'informatique septembre 1988 pp 205 à 222

Leclercq D., *Les grands types d'apprentissage selon R. M. Gagné*, in Éducation n° 137 – septembre-octobre 1972

Leclercq D., *Les grands types d'apprentissage selon R. M. Gagné (II). Critique des catégories supérieures du modèle de Gagné*, in Éducation n° 138 – novembre-décembre 1972

Linn M. C., Dalbey J., *Cognitive consequences of programming instruction* in Studying the Novice Programmer, University of Michigan 1989 PP 57 à 81

Meurice de Dormale R., *Robotique virtuelle* in Les actes de la 5ème rencontre francophone sur la didactique de l'informatique, Monastir 1996

Meurice de Dormale R., *Robotique virtuelle : un environnement pour un apprentissage dynamique de l'algorithmique* in Les actes du 5ème colloque international sur la robotique pédagogique, Montréal 1997

Putnam R. T., Sleeman D., Baxter J. a. and Kuspa L. K., *A Summary of Misconceptions of High-School BASIC programmers* in Studying the Novice Programmer, University of Michigan 1989 pp 301 à 314

Romainville M., *Une analyse critique de l'initiation à l'informatique : quels apprentissages et quels transferts ?* EPI Colloque francophone sur la didactique de l'informatique septembre 1988 pp 225 à 241

Spohrer J. C., Soloway E., *Novice mistake : are the folk wisdomscorrect ?* in Studying the Novice Programmer, University of Michigan 1989 pp 401 à 416

Table des matières